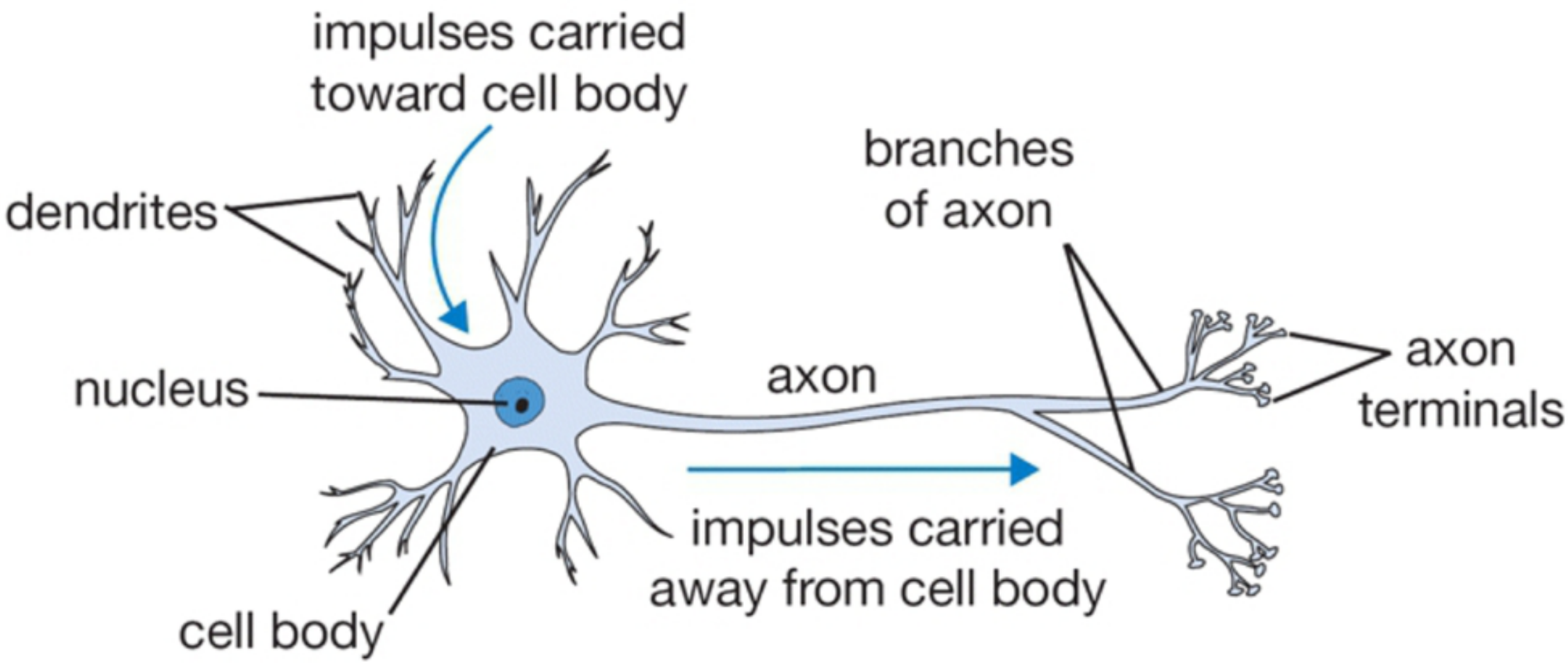
Introduction to Neural Networks

If not stated otherwise graphs and code examples were taken with permission from Michael Nielsen's online book Neural Networks and Deep Learning.

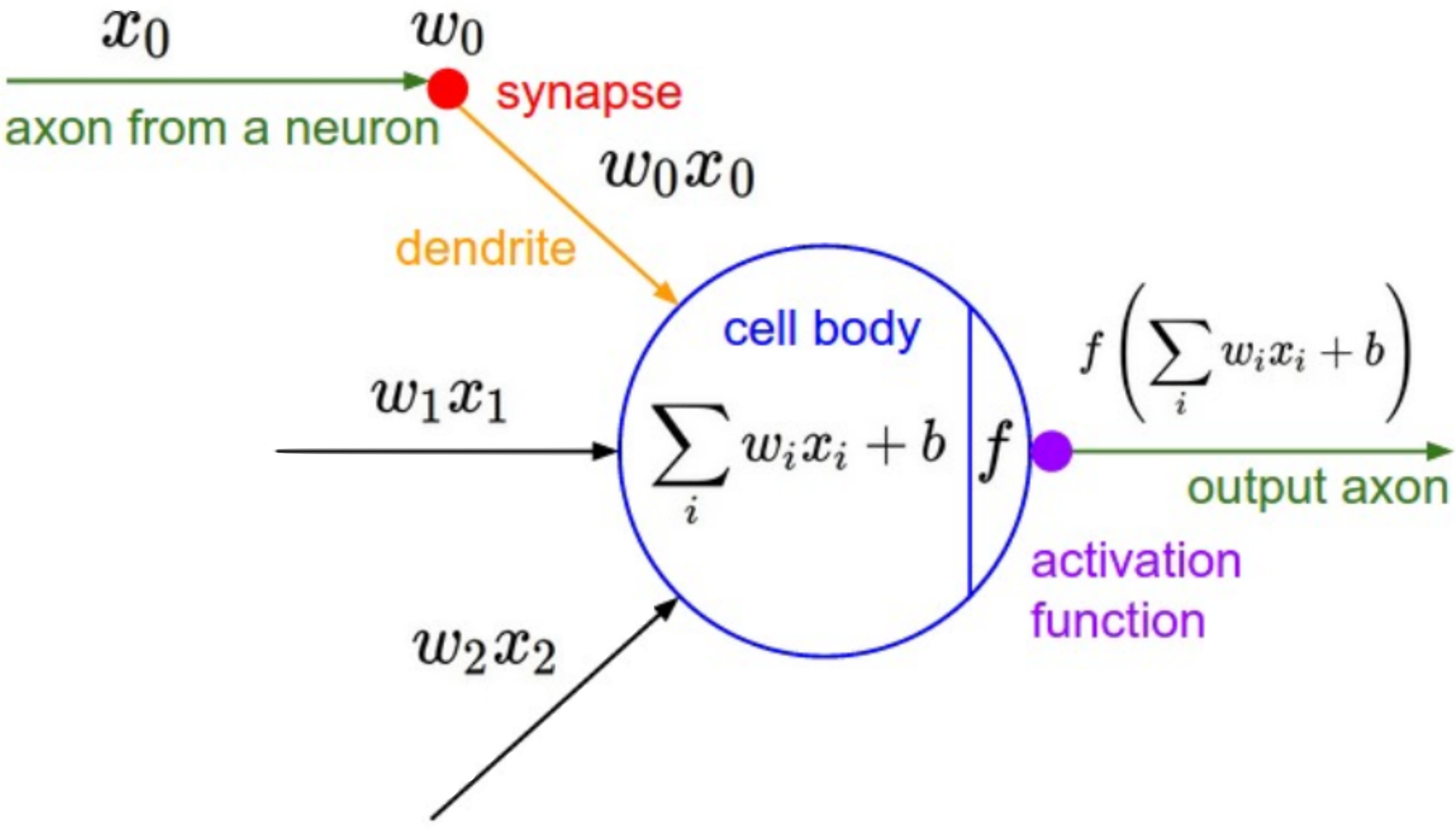

5 0 4 1 9 2

A neuron



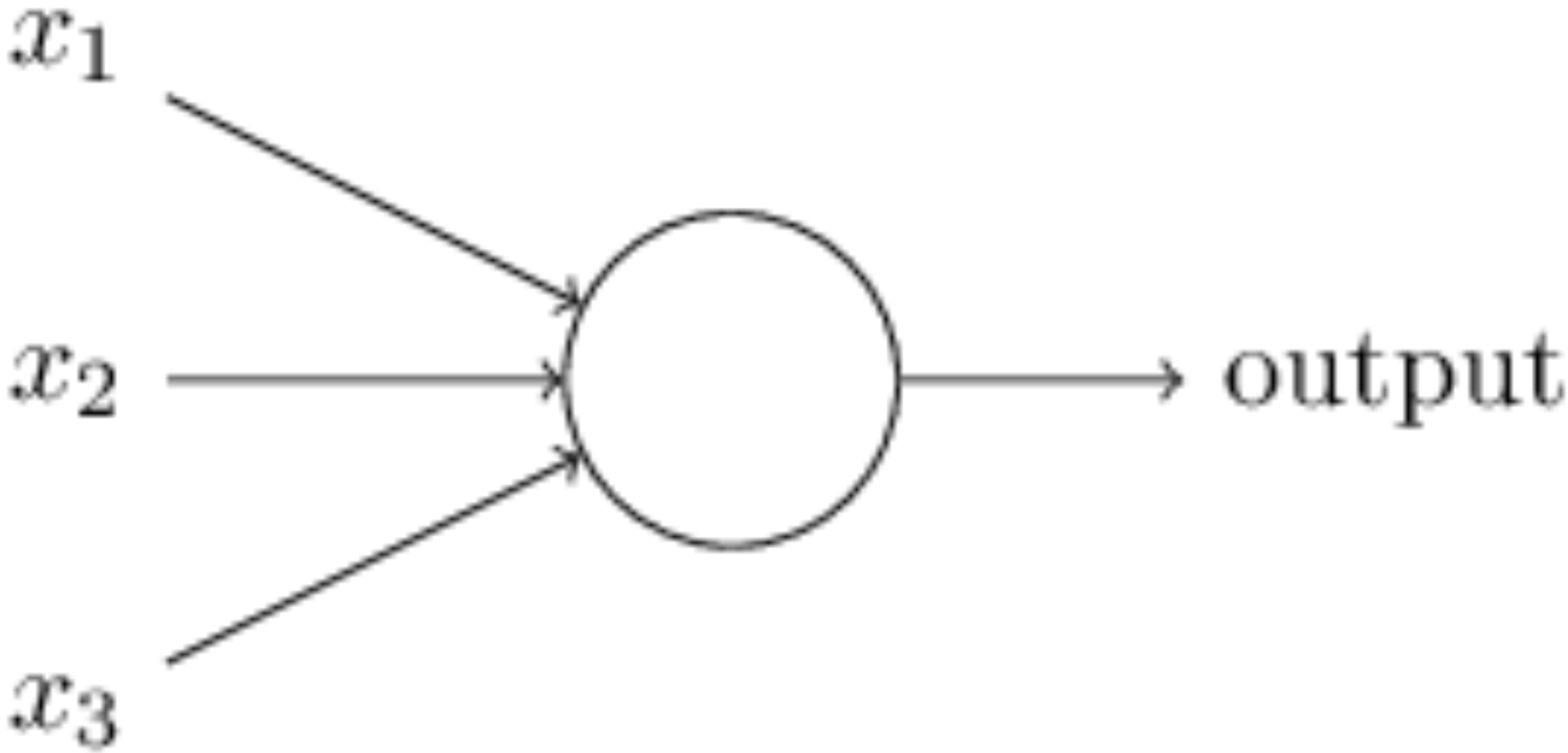
http://cs231n.github.io/neural-networks-1/

Artificial Neuron

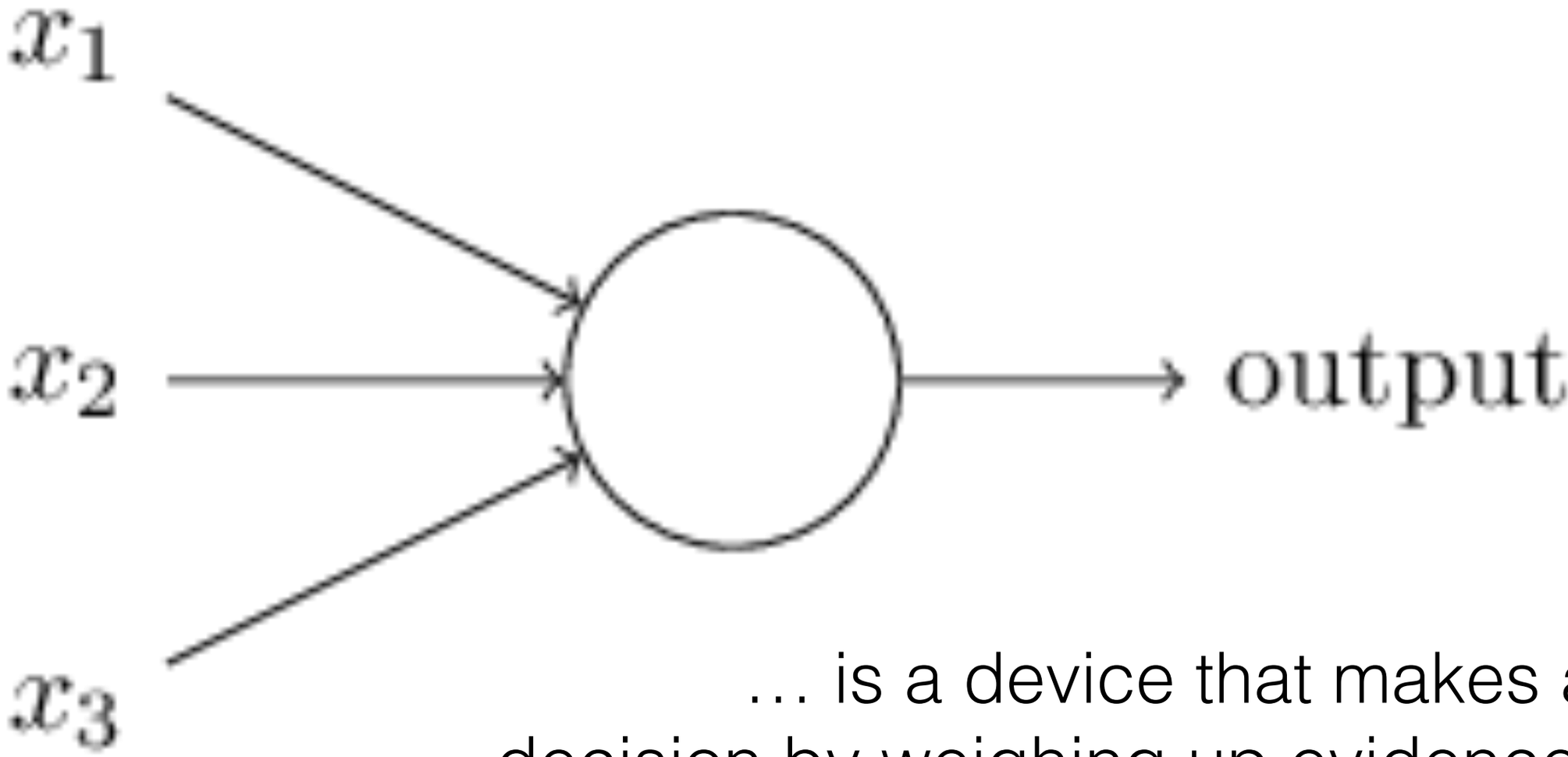


cs231n.github.io/neural-networks-1/

The Perceptron (1957)

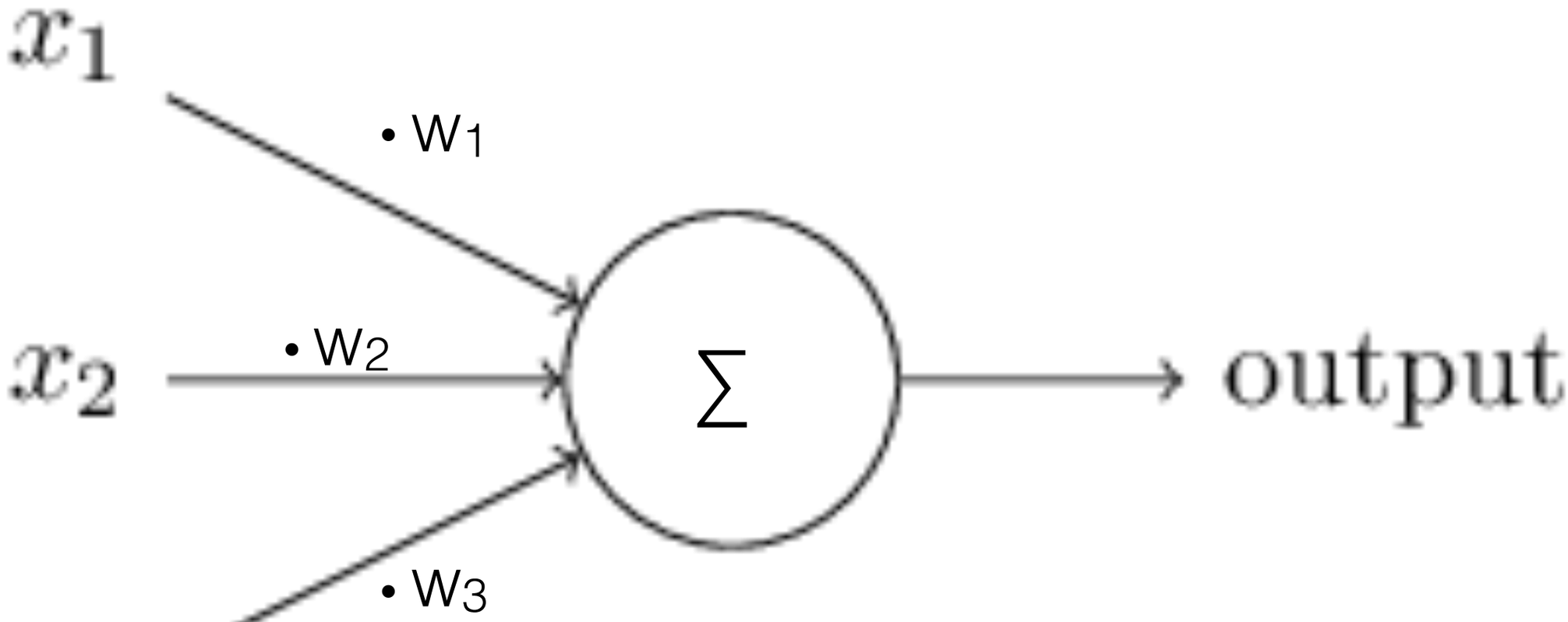


The Perceptron (1957)



… is a device that makes a decision by weighing up evidence.

The Perceptron (1957)



$$\text{output} = \begin{cases} 0 & \text{if } \sum_j w_j x_j \leq \text{ threshold} \\ 1 & \text{if } \sum_j w_j x_j > \text{ threshold} \end{cases}$$

Notation

$$\text{output} = \begin{cases} 0 & \text{if } \sum_j w_j x_j \leq \text{ threshold} \\ 1 & \text{if } \sum_j w_j x_j > \text{ threshold} \end{cases}$$

$$\text{output} = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases}$$

dot product

threshold = -**b**ias

Exercise

Making decision with a perceptron

- **Setting:** Next Sunday there is a cheese festival in town.
- **Qu**
- **Pa**
  - How is the weather?
  - Are my friends coming?
  - Will I have a hangover from the night before?
- **Notebook:**
  - `~/Perceptron.ipynb`

Launching the ec2 instance:

http://bit.ly/1sYCMNh

Exercise

Making decision with a perceptron

- **Setting:** Next Sunday there is a cheese festival in town.
- **Question:** Should I go or not?
- **Parameters:**
  - How is the weather?
  - Are my friends coming?
  - Will I have a hangover from the night before?
- **Notebook:**
  - `~/Perceptron.ipynb`

But where do the weights come from?

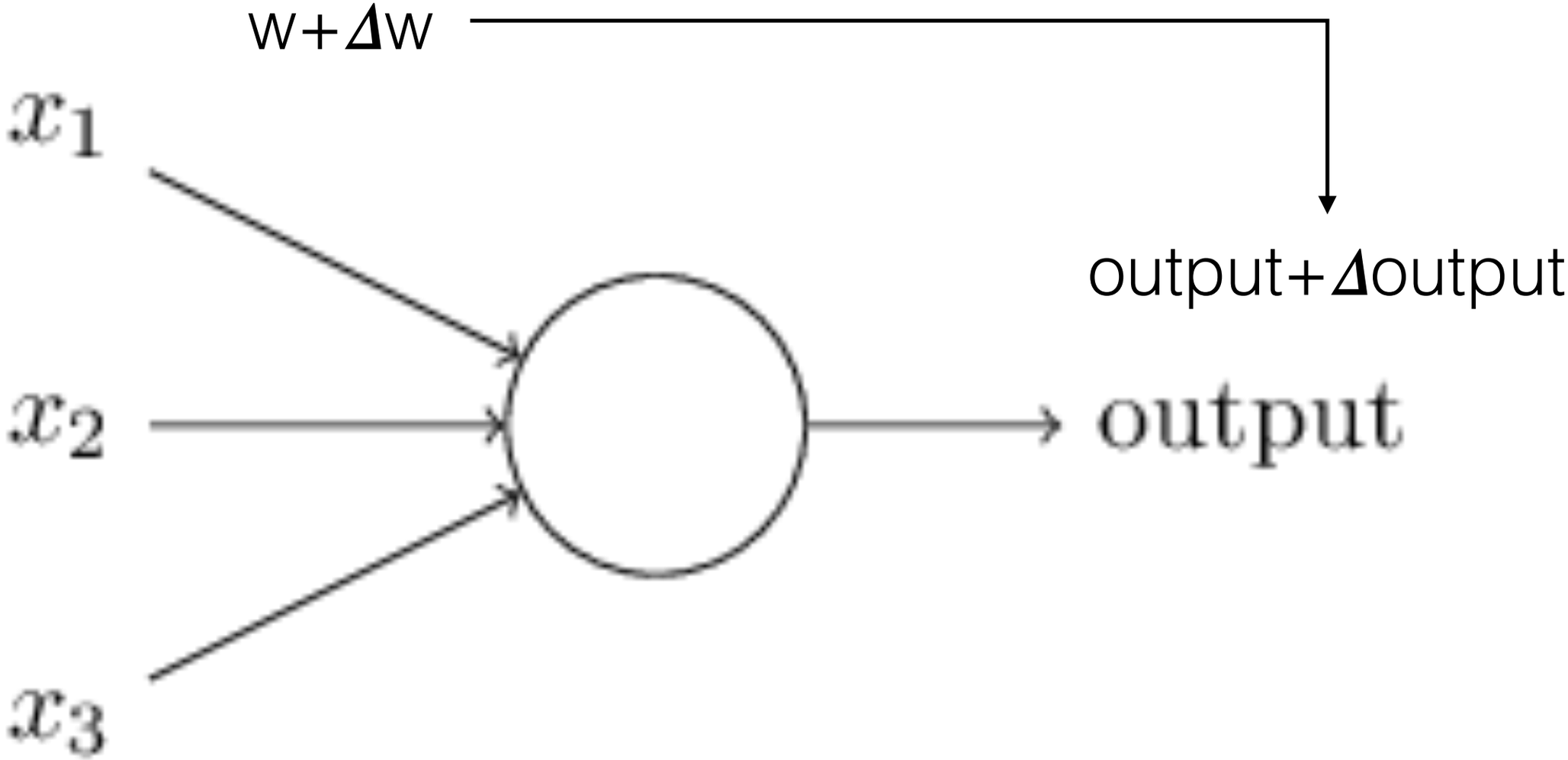
Random?

God mode?

Intelligent trial-and-error

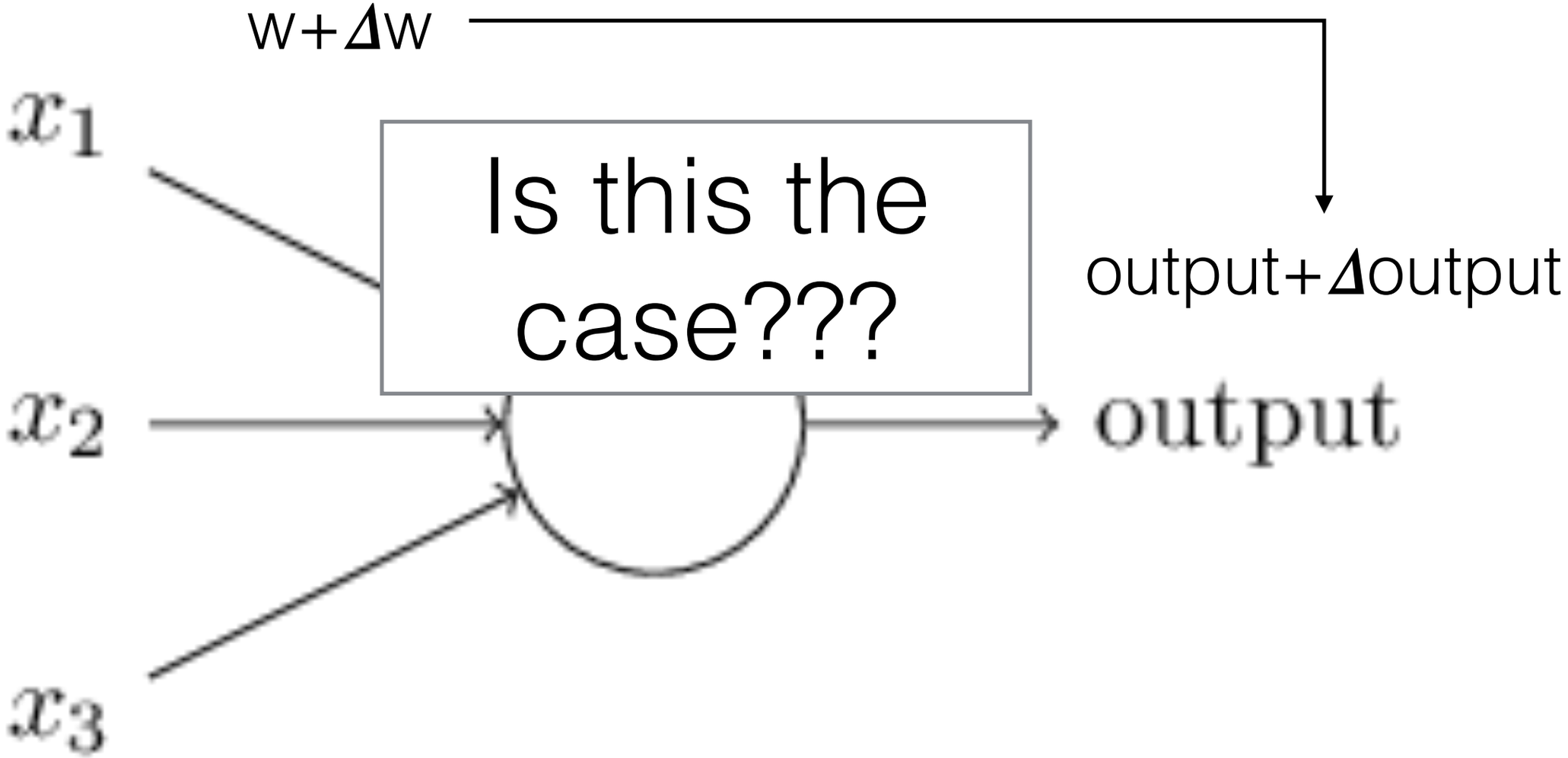
What do we want?

A small change in the weights (or biases) should only cause a small change in the output.

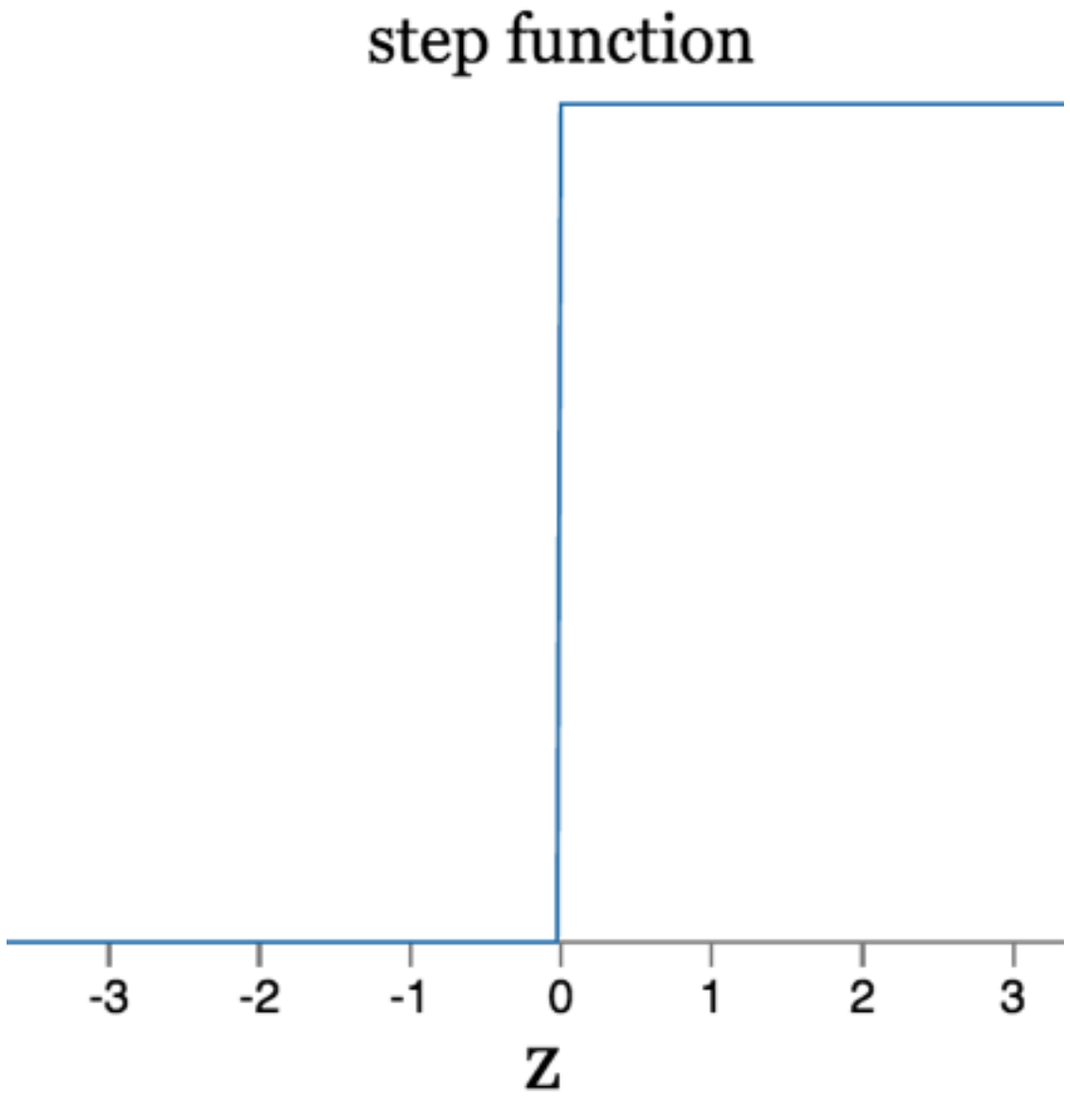


What do we want?

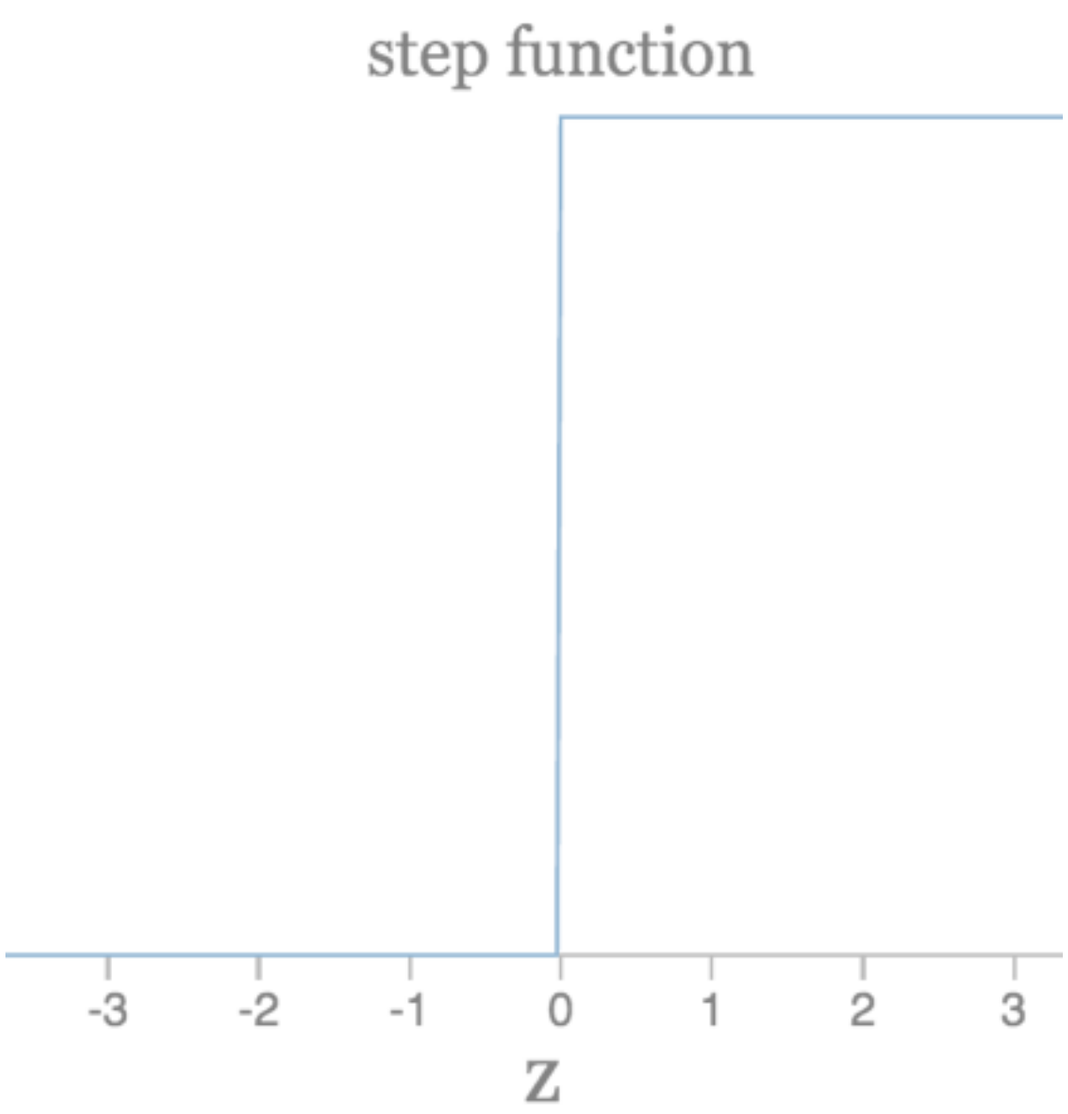
A small change in the weights (or biases) should only cause a small change in the output.

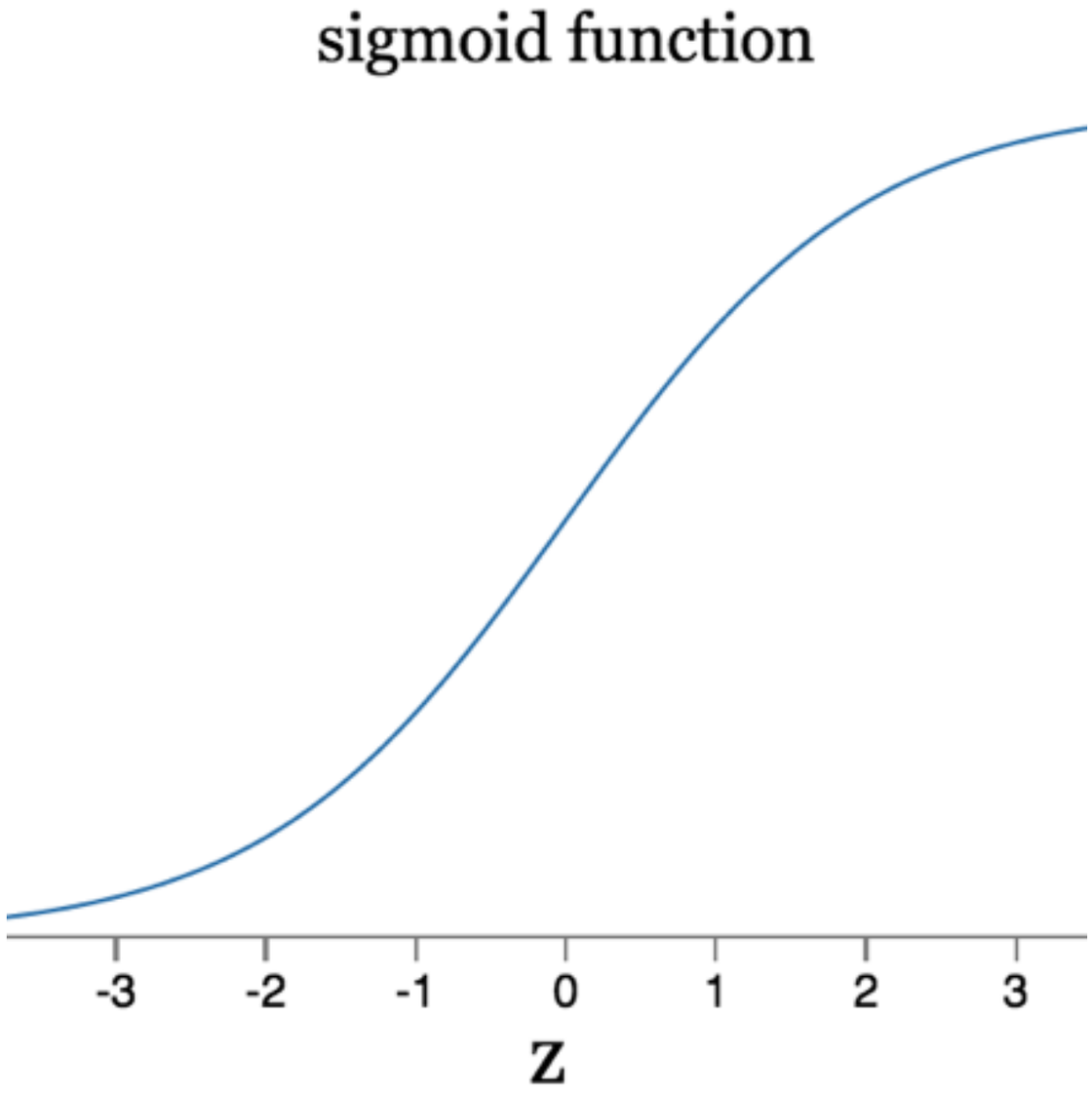


Activation functions



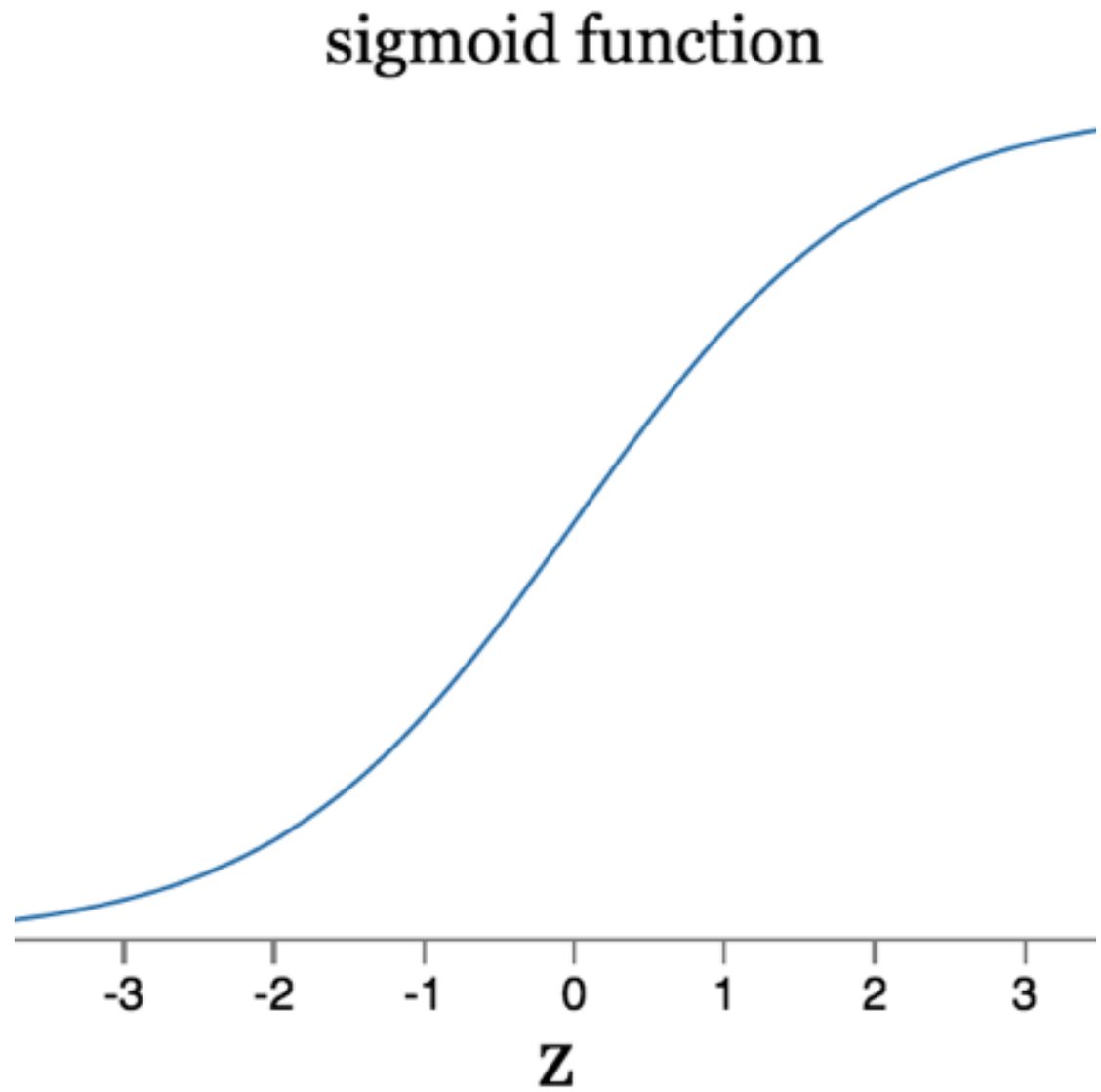
Activation functions



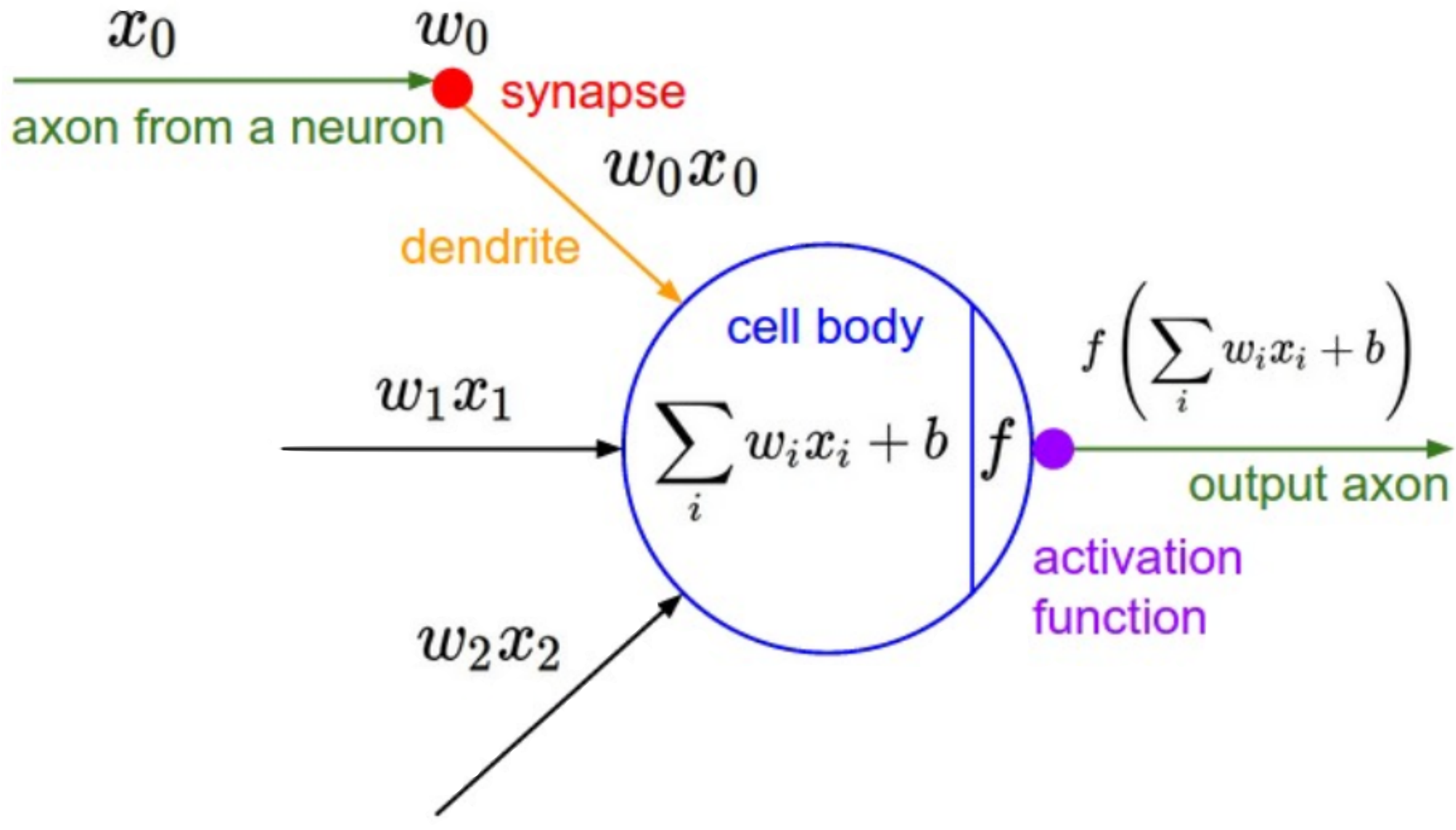


Activation functions

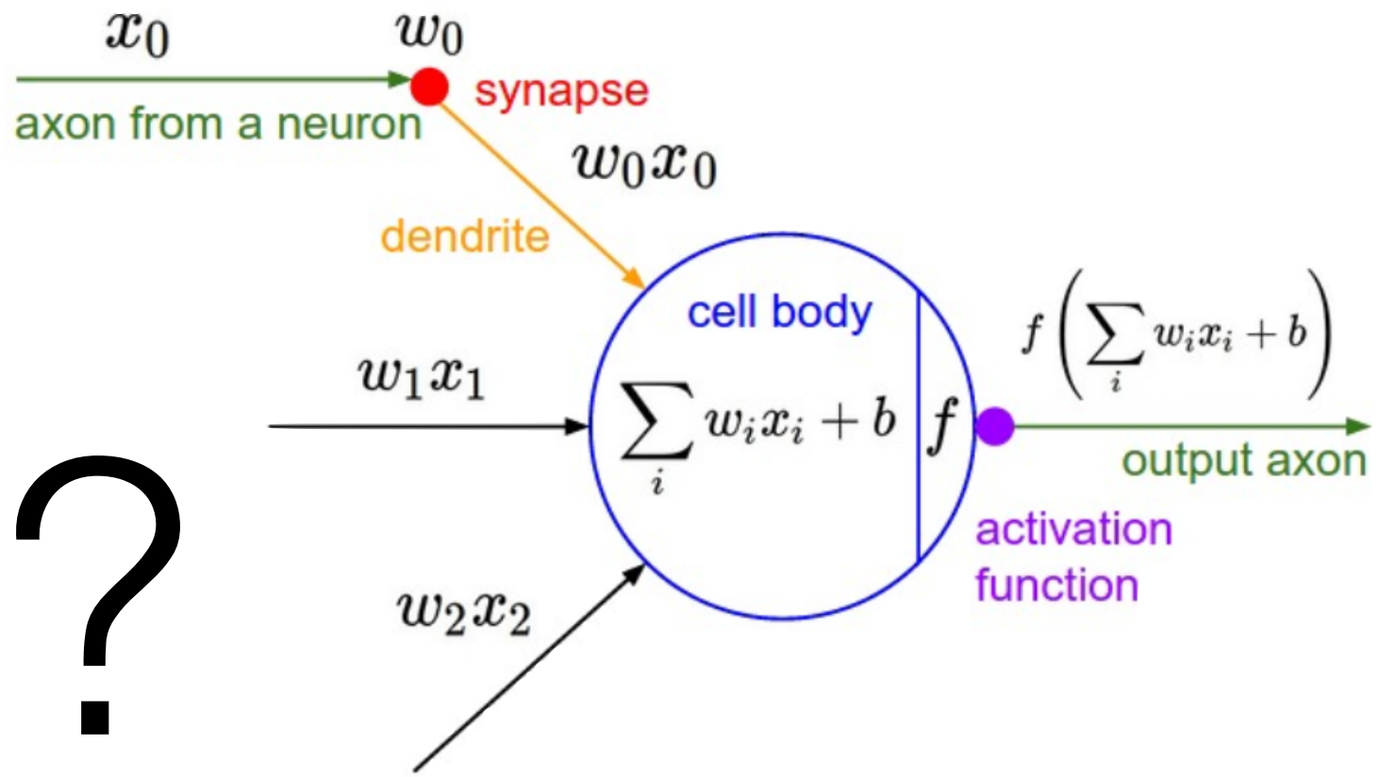
$$\sigma(z) \equiv \frac{1}{1+e^{-z}}.$$



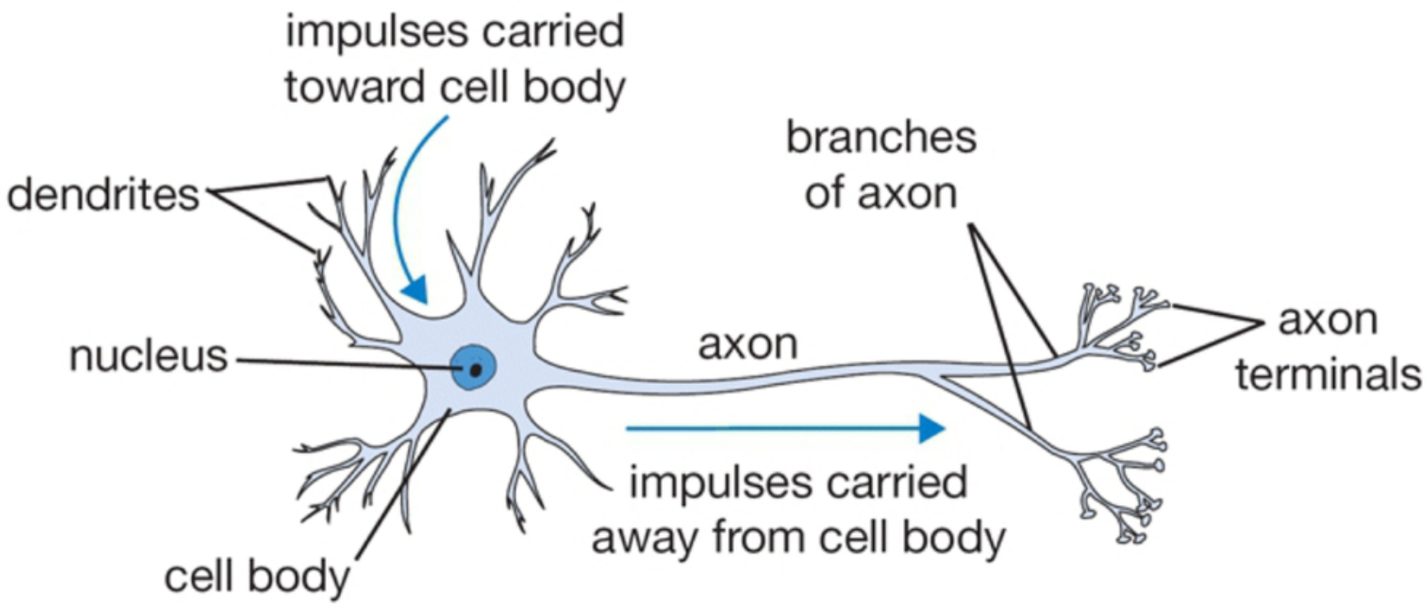
Artificial Neuron



cs231n.github.io/neural-networks-1/

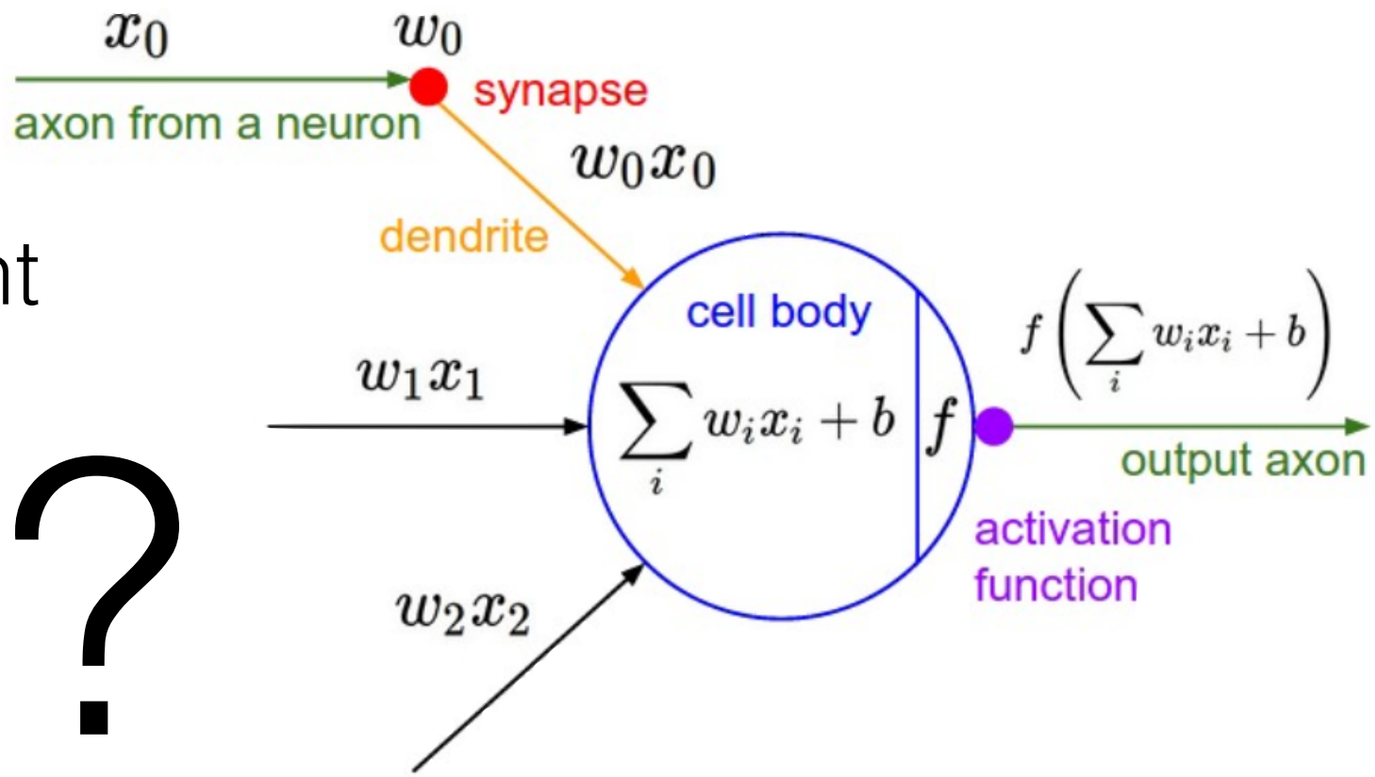


?

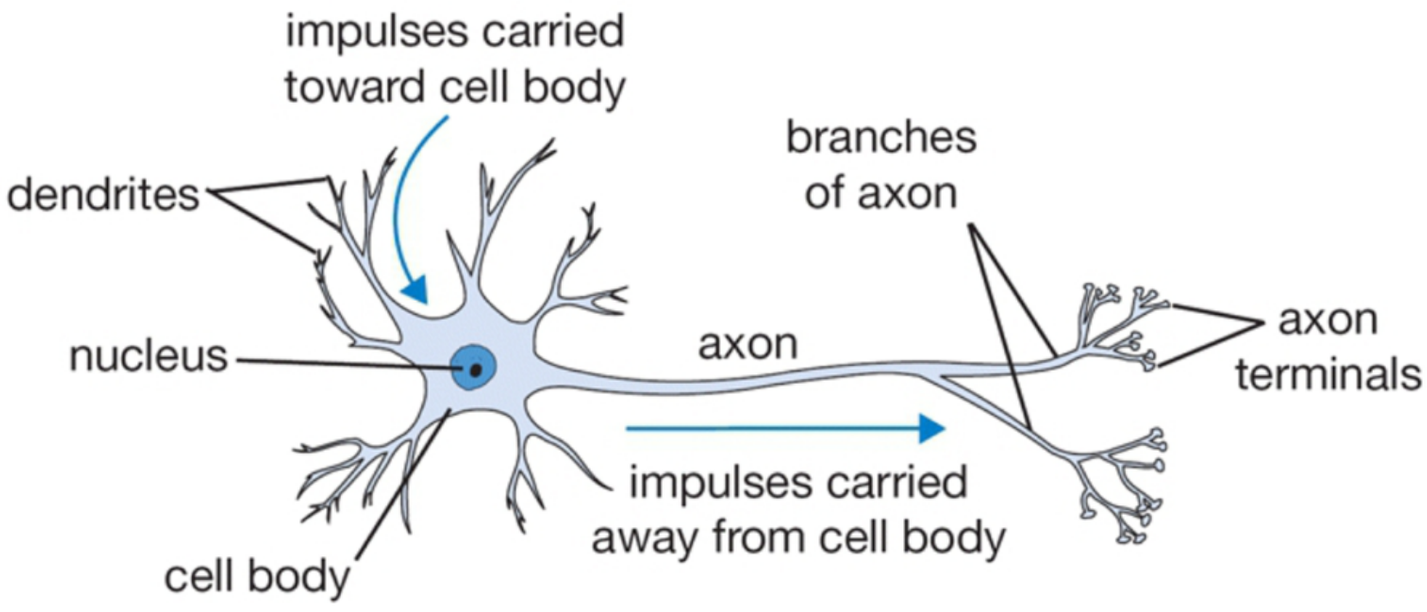


http://cs231n.github.io/neural-networks-1/

- There are many different types of neurons
- The dendrites perform complex non linear operations



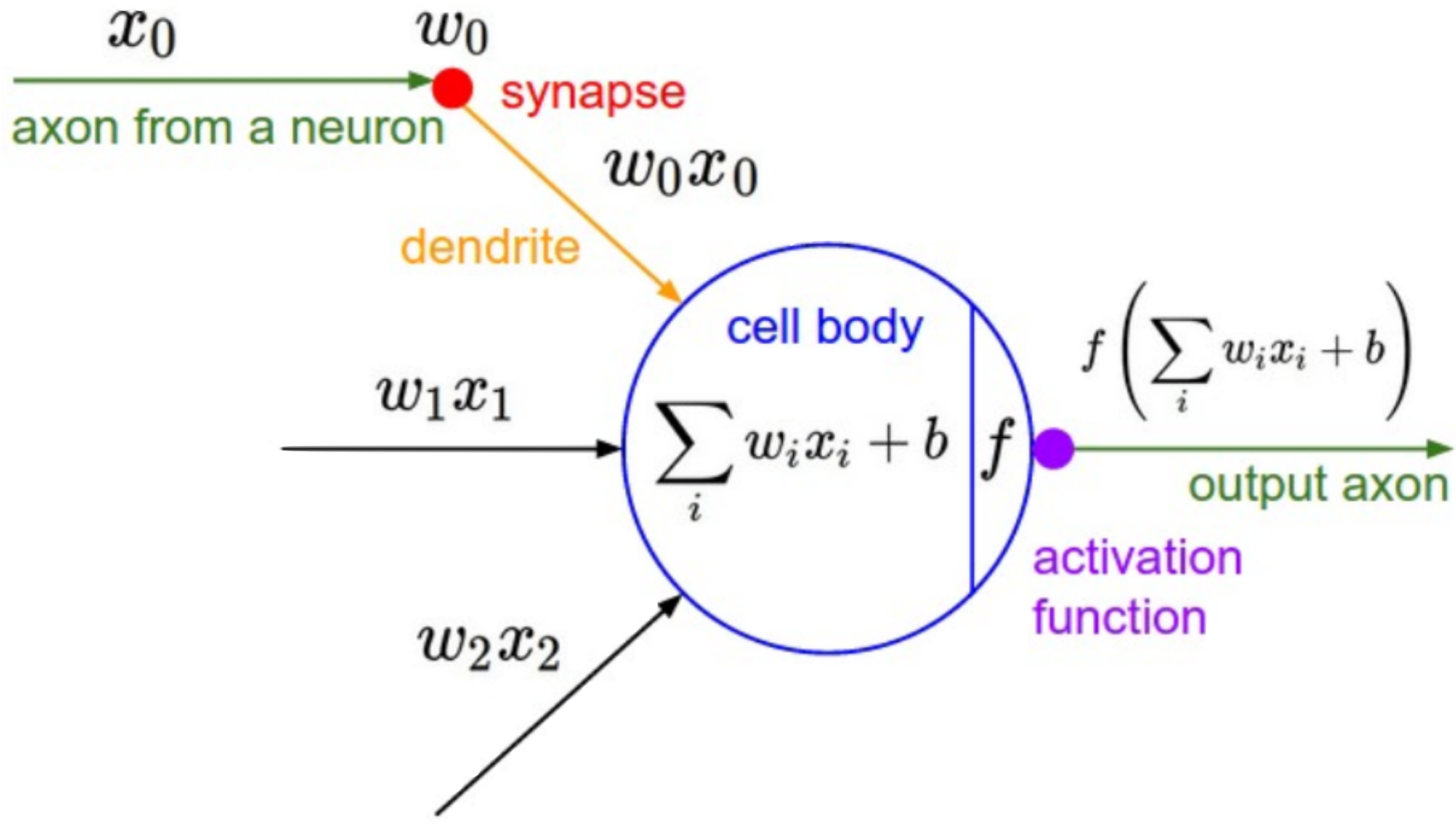
?



- The output activations are *spikes* not a constant number
- The timing of spikes significant

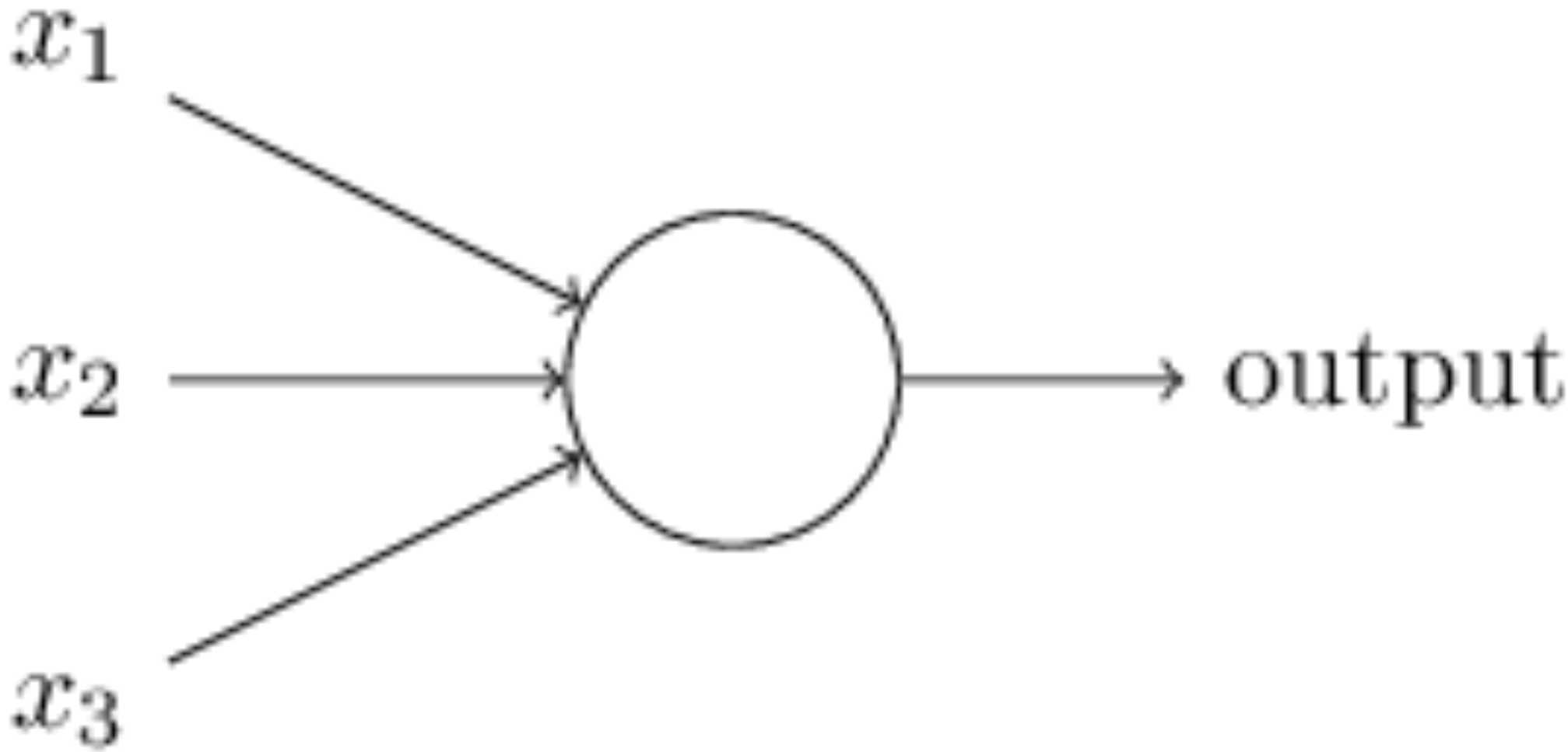
http://cs231n.github.io/neural-networks-1/

Artificial Neuron



cs231n.github.io/neural-networks-1/

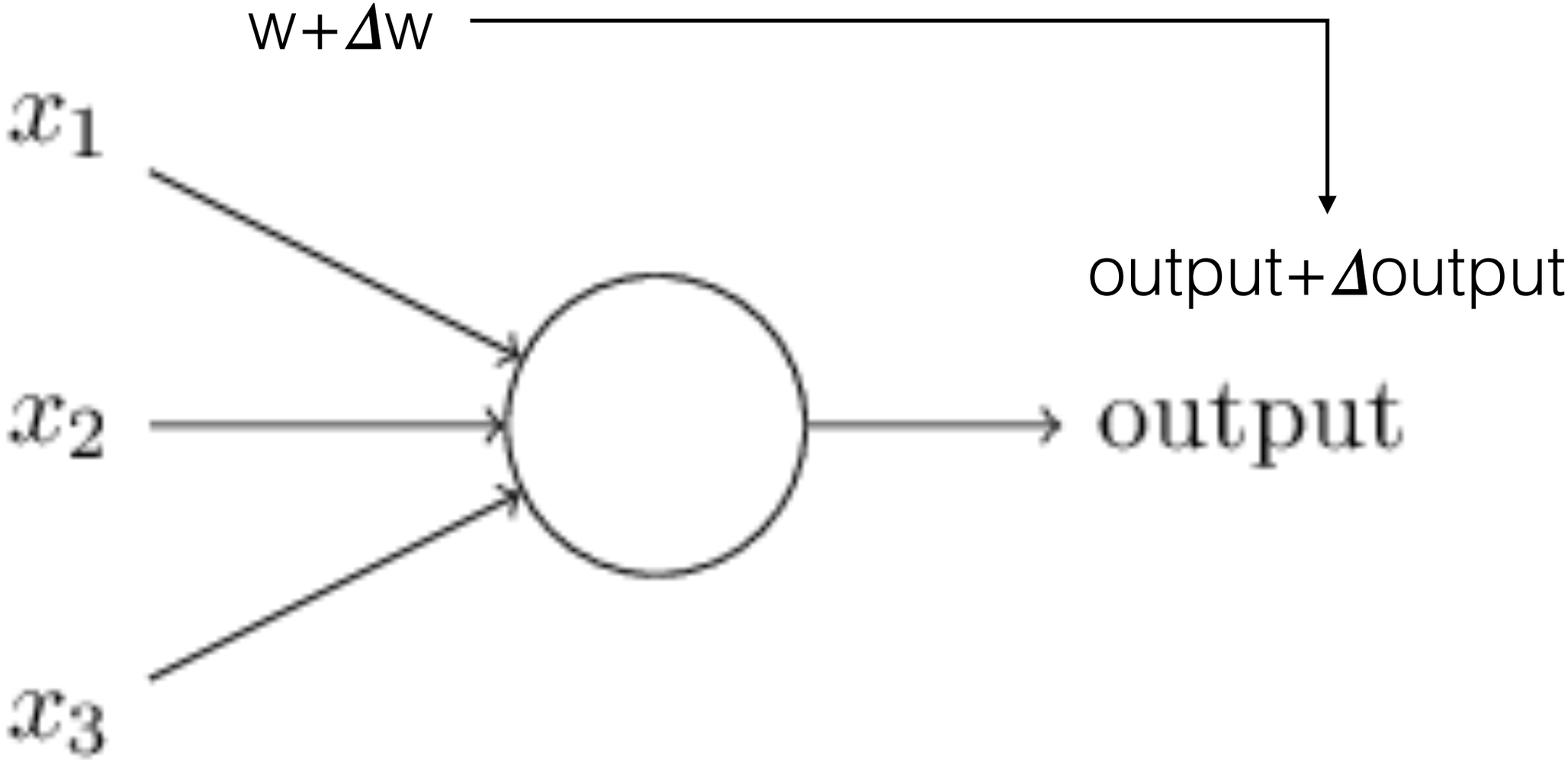
The Sigmoid neuron (80s)



Back to learning the weights

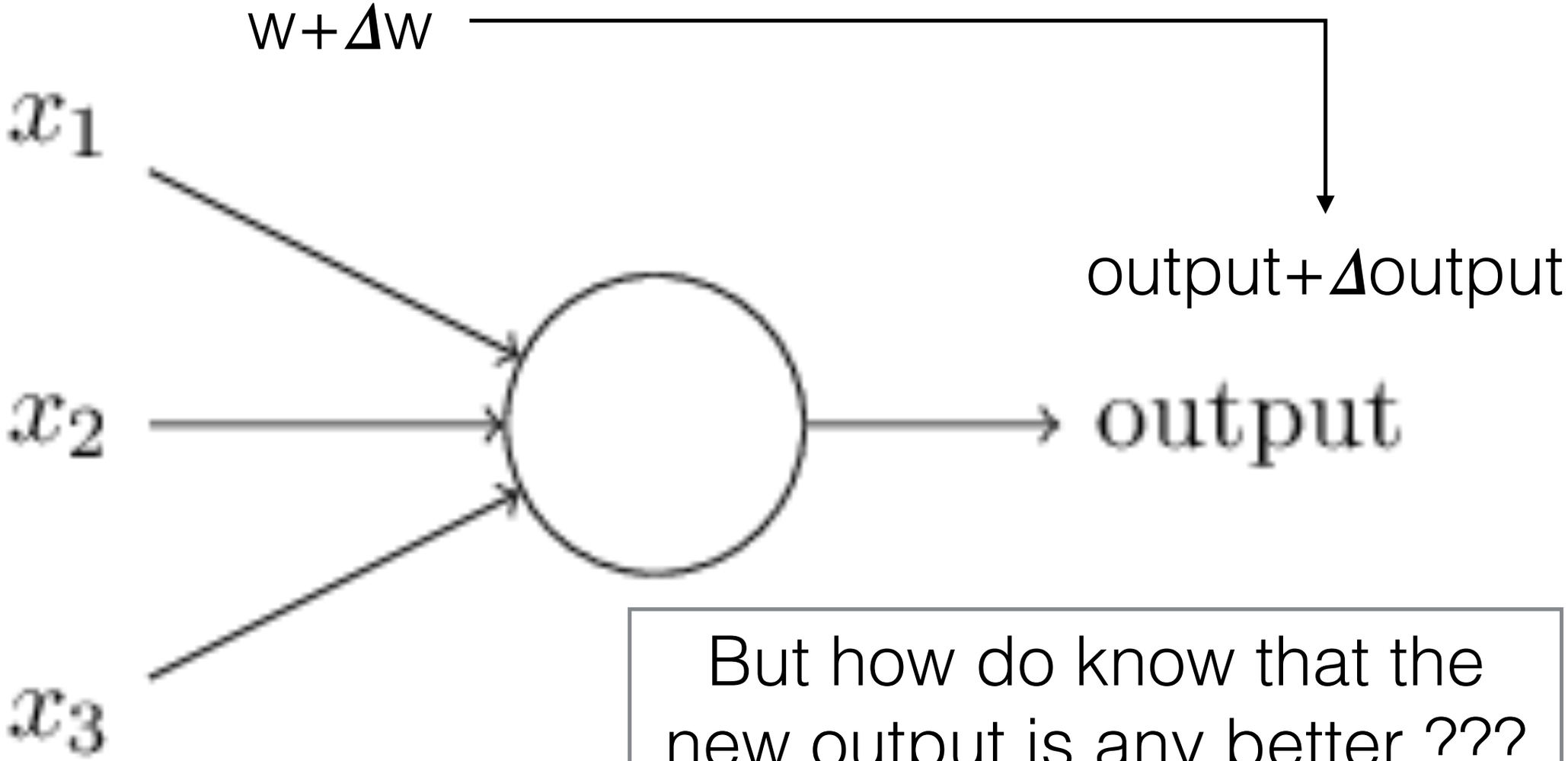
What do we want?

A small change in the weights (or biases) should only cause a small change in the output.



What do we want?

A small change in the weights (or biases) should only cause a small change in the output.



But how do know that the new output is any better ???

Loss functions

Also cost or objective functions

Mean Squared Error (MSE)

An example cost function

$$C(w,b) \equiv \frac{1}{2n} \sum_x \|y(x) - a\|^2$$

C = cost function (here Mean squared error)
w = weights
b = bias

x = input
y(x) = desired output
a = output
||v|| = length of vector
n = number of sample

Why so complicated?

Why not just counting the correct predictions?

Why so complicated?

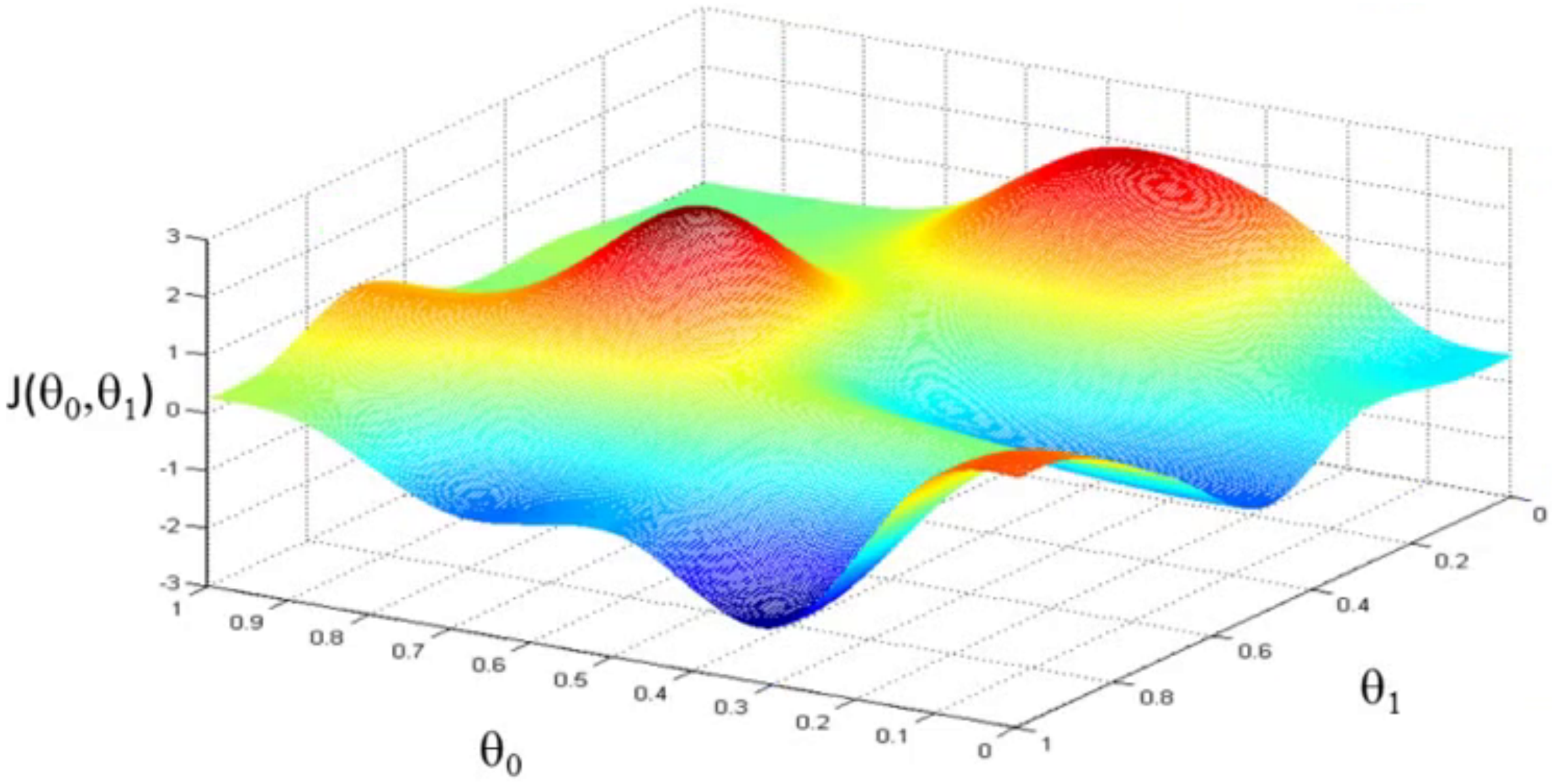
Why not just counting the correct predictions?

What else can loss functions do for us?

What do we want to do with the cost function?

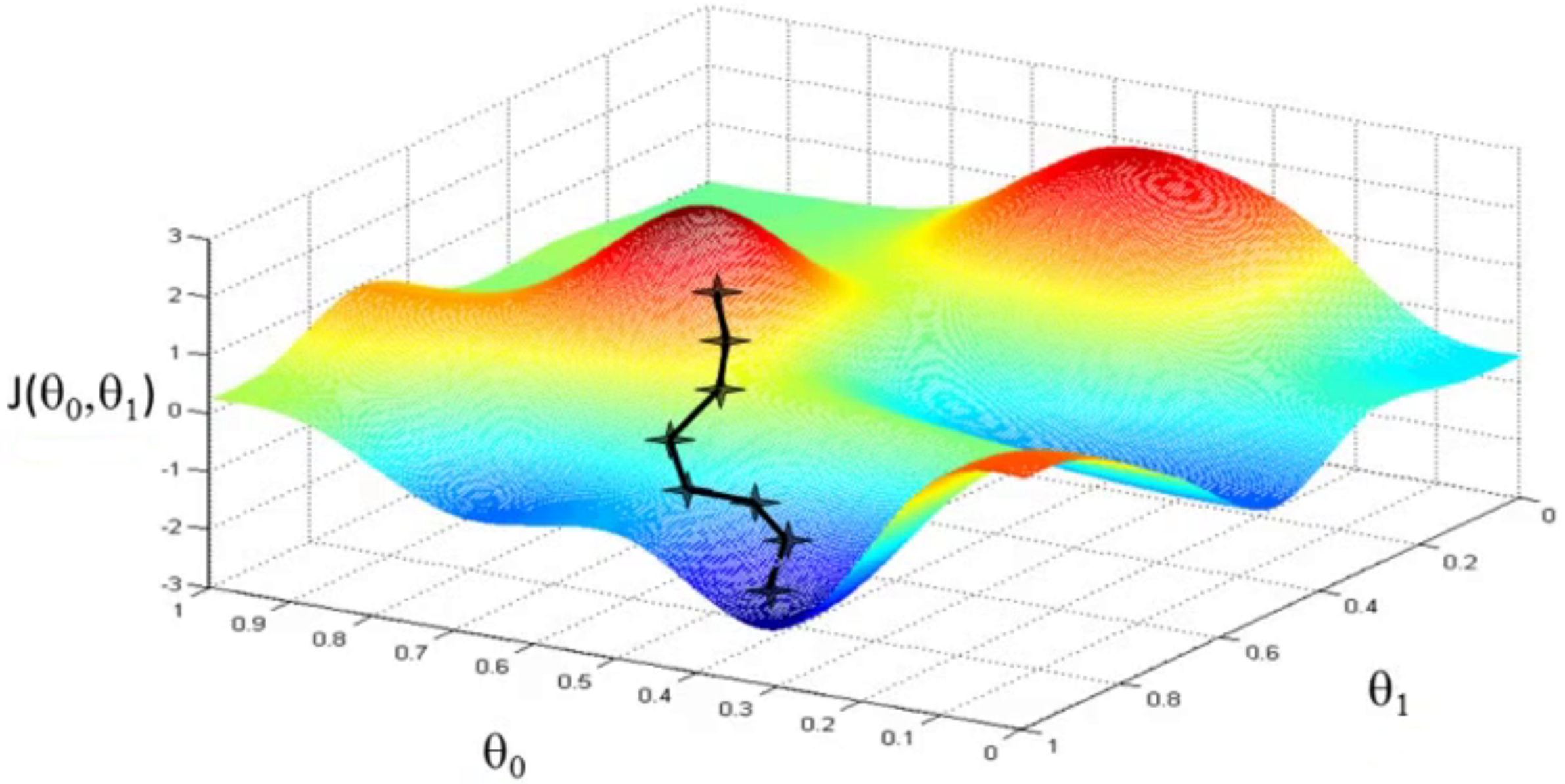
Why not use calculus to compute it the minimum?

Gradient Descent



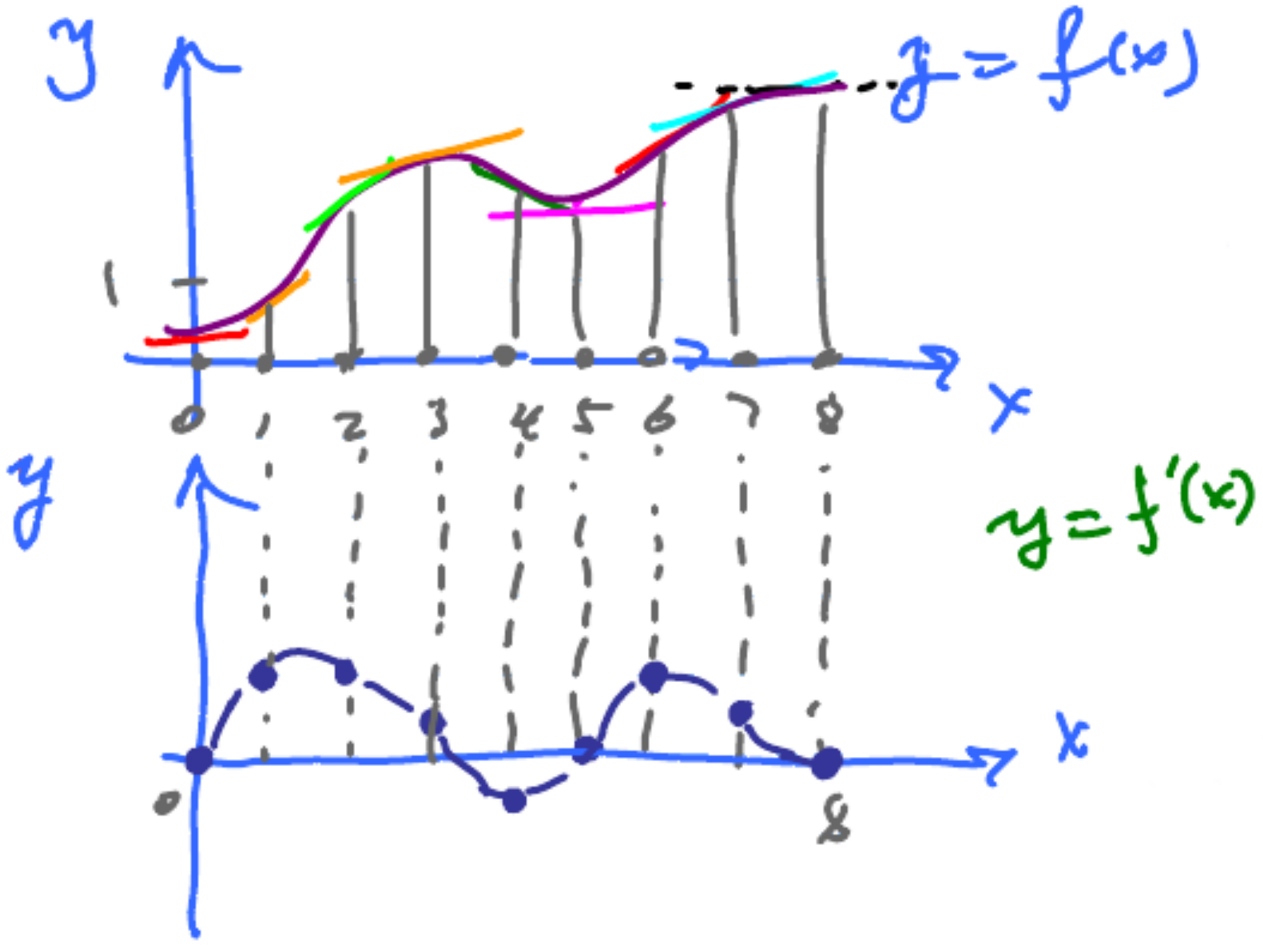
https://class.coursera.org/ml-003/lecture

Gradient Descent



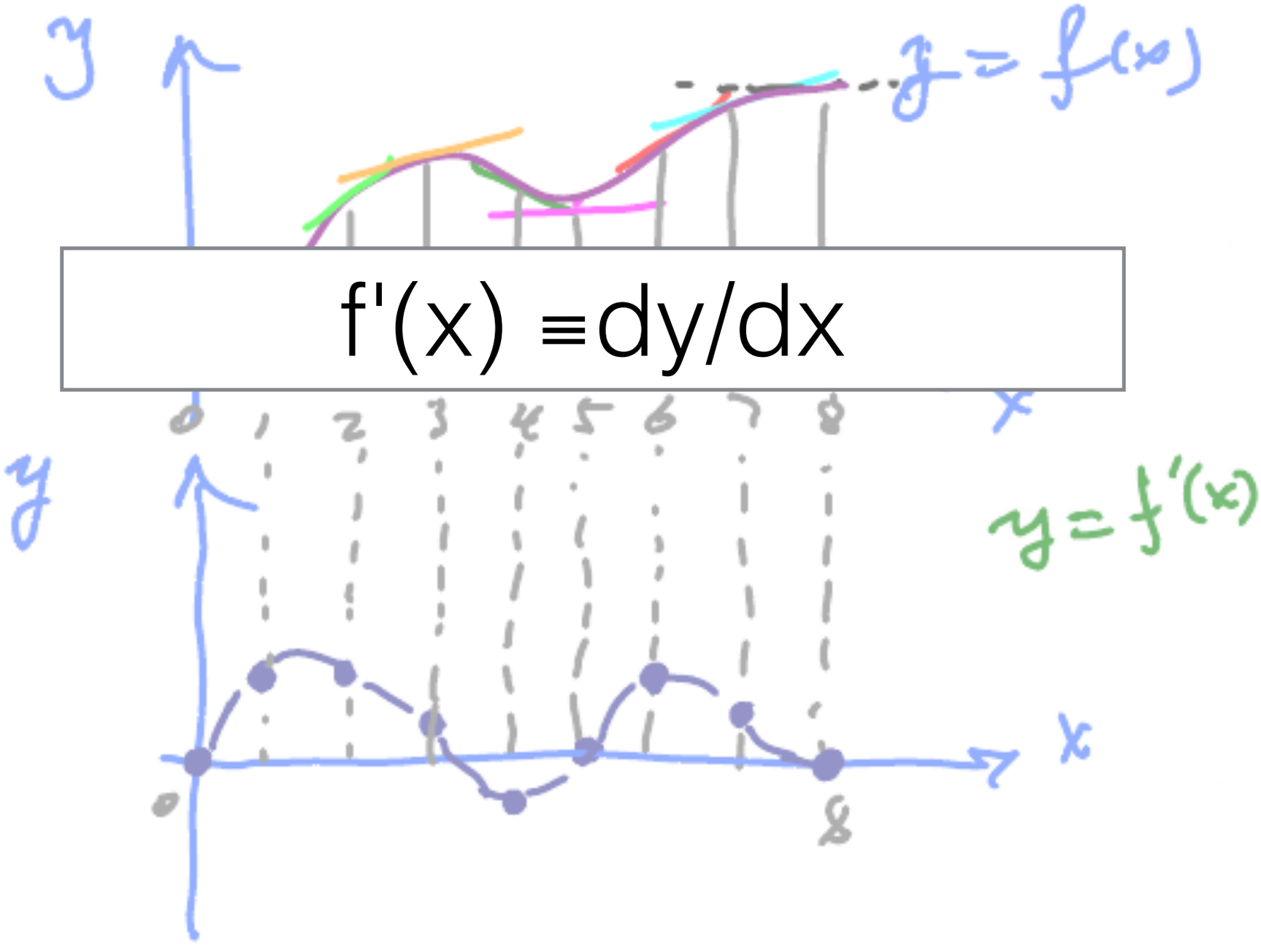
https://class.coursera.org/ml-003/lecture

What is a derivative?



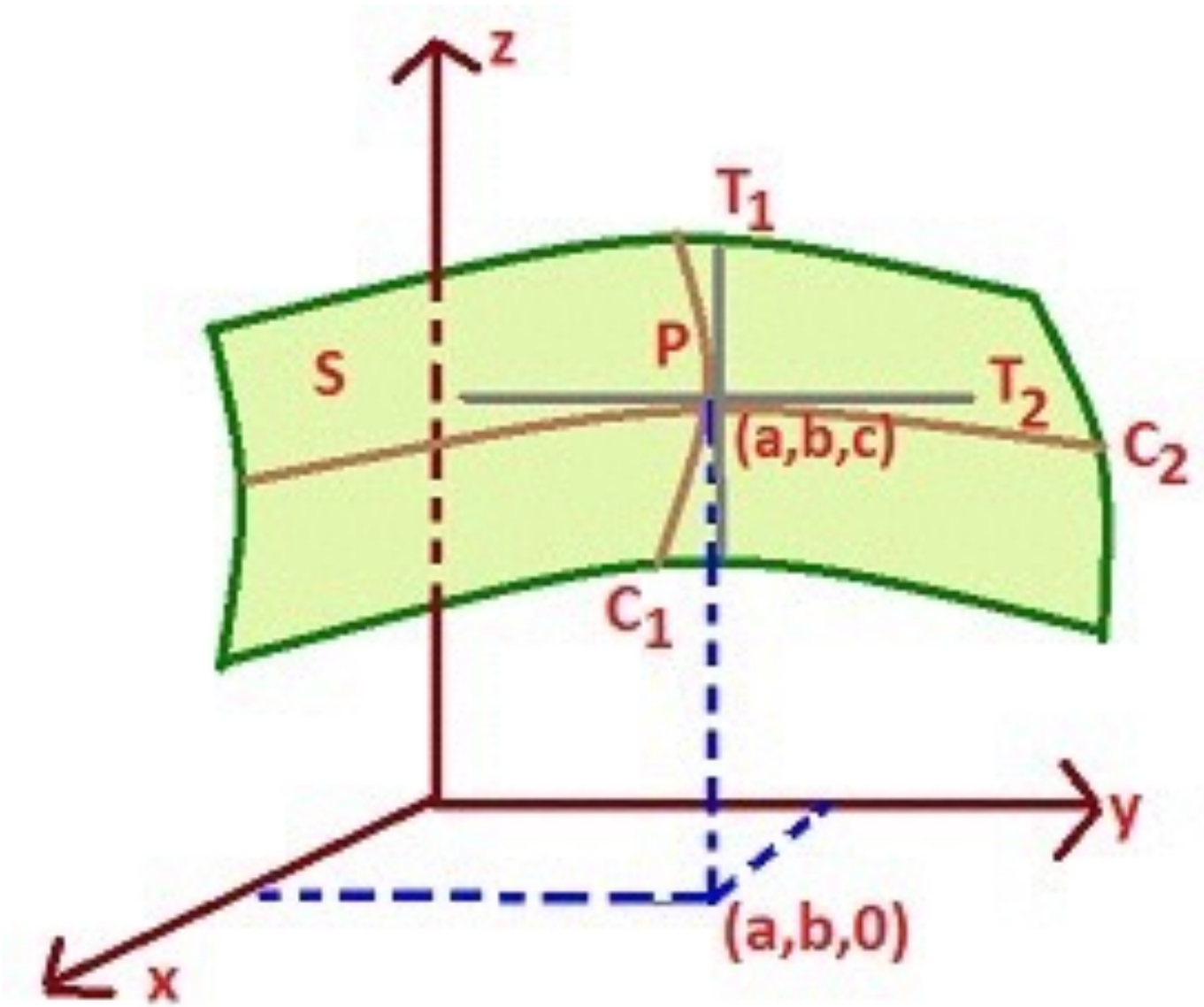
http://inperc.com/wiki/index.php?title=Derivative_as_a_function

What is a derivative?



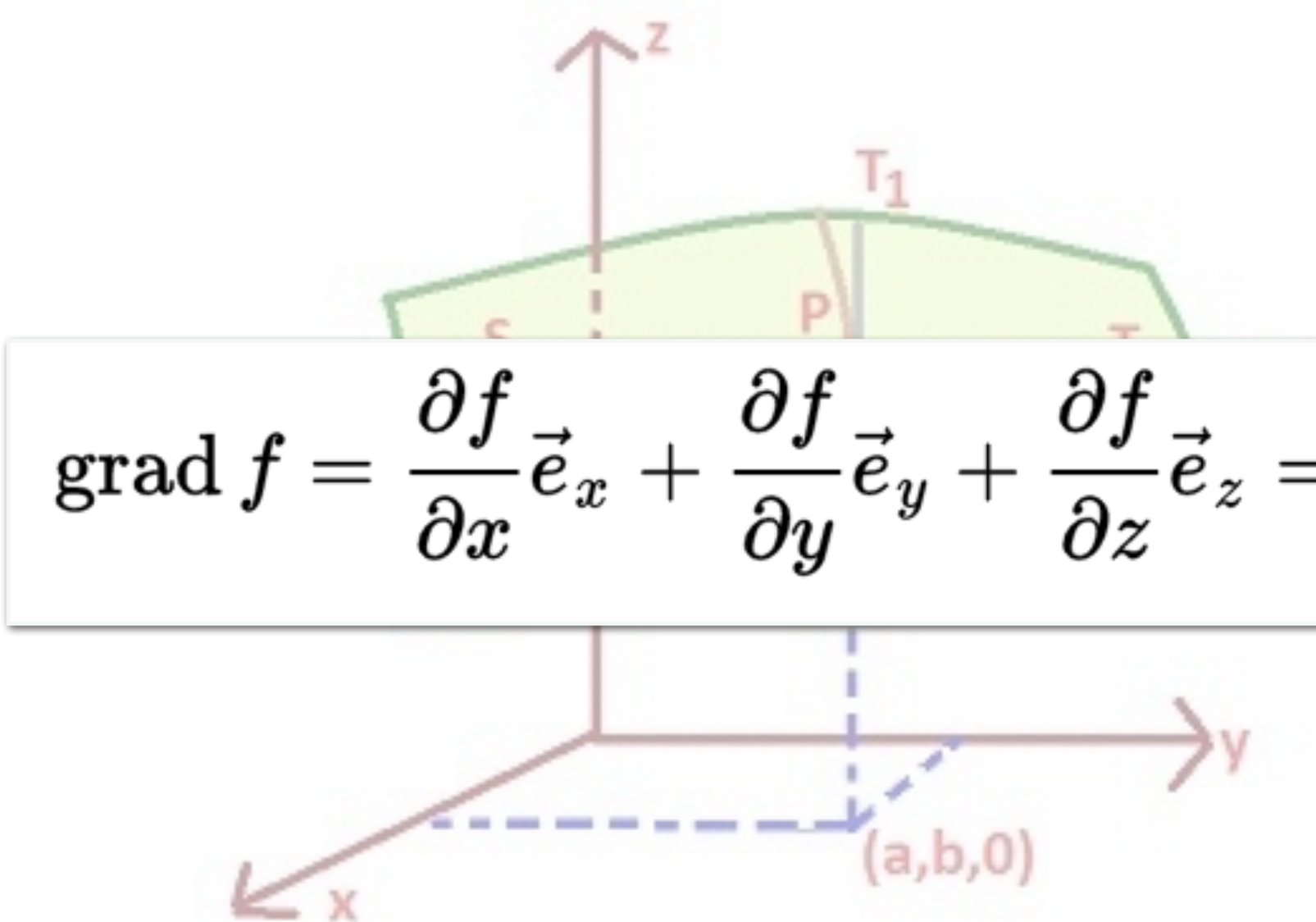
http://inperc.com/wiki/index.php?title=Derivative_as_a_function

Partial derivatives???



http://www.mathcaptain.com/calculus/partial-derivative.html

Partial derivatives???



$$\operatorname{grad} f = \frac{\partial f}{\partial x}\vec{e}_x + \frac{\partial f}{\partial y}\vec{e}_y + \frac{\partial f}{\partial z}\vec{e}_z = \nabla f$$

$\boldsymbol{e_i}$ are the orthogonal unit vectors pointing in the coordinate directions

http://www.mathcaptain.com/calculus/partial-derivative.html

Updating weights using Gradient Descent

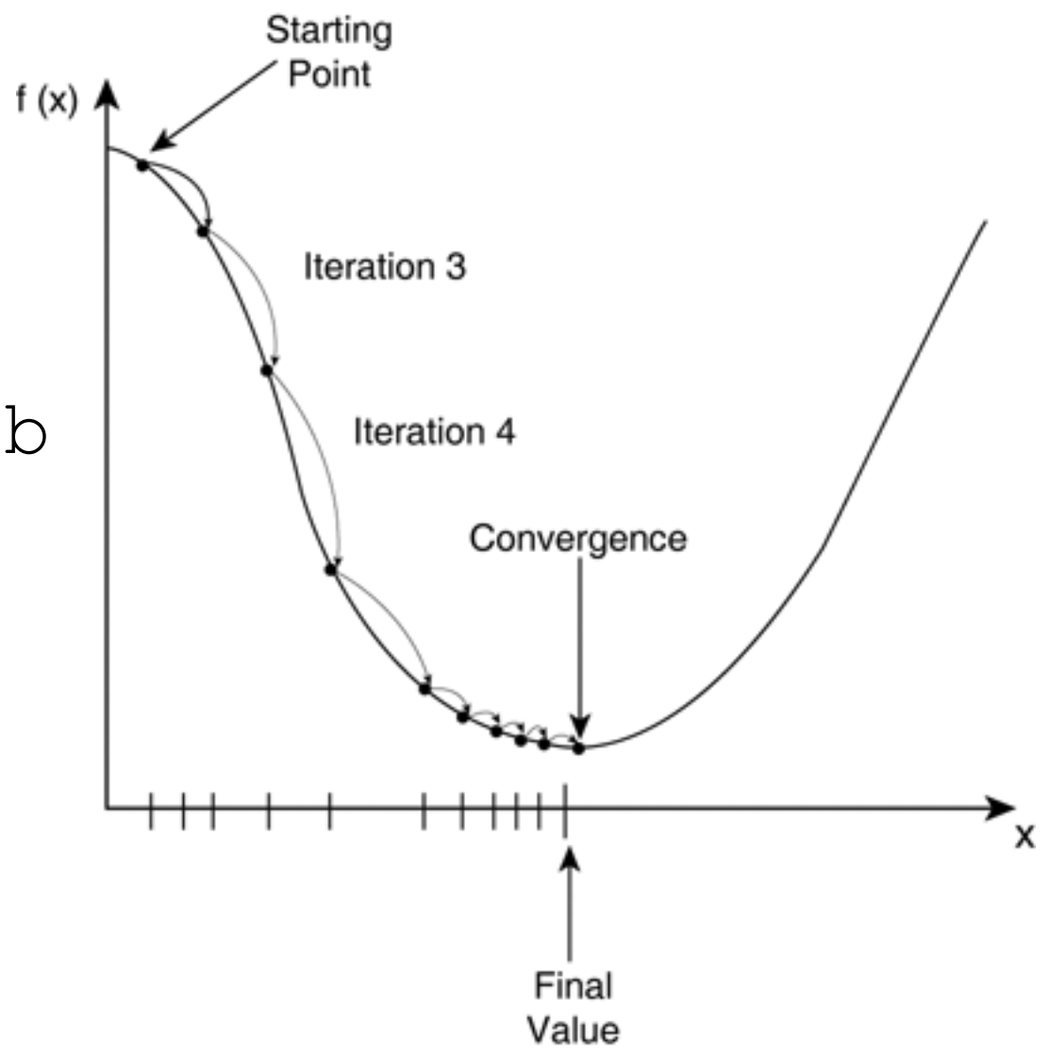
$$w \rightarrow w' = w - \eta \nabla C$$

∇C = gradient of C
w = old weight
w'= new weight
η = learning rate (greek eta)

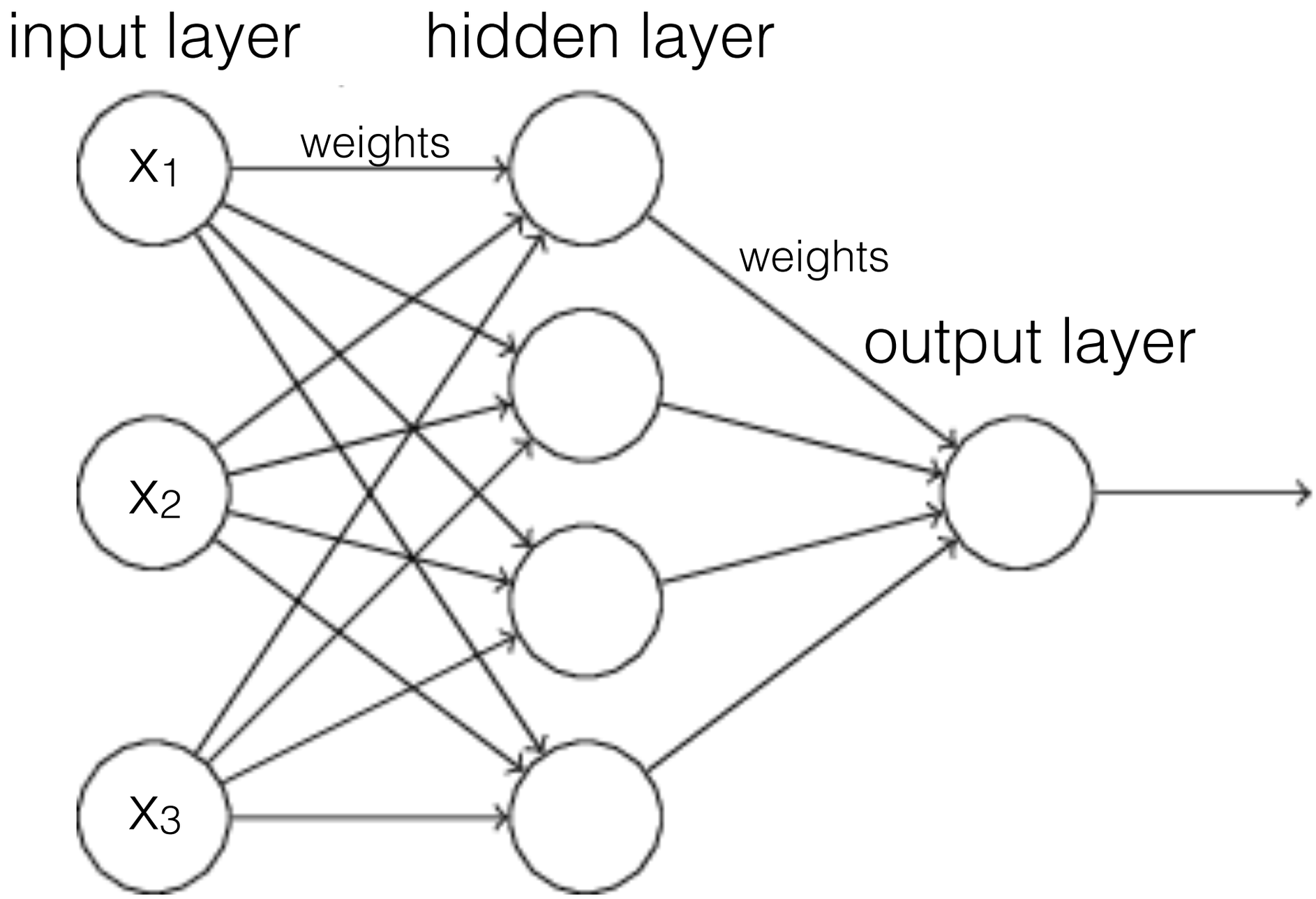
Exercise gradient descent

- Use f(x) = x**2 as function
- Notebook:

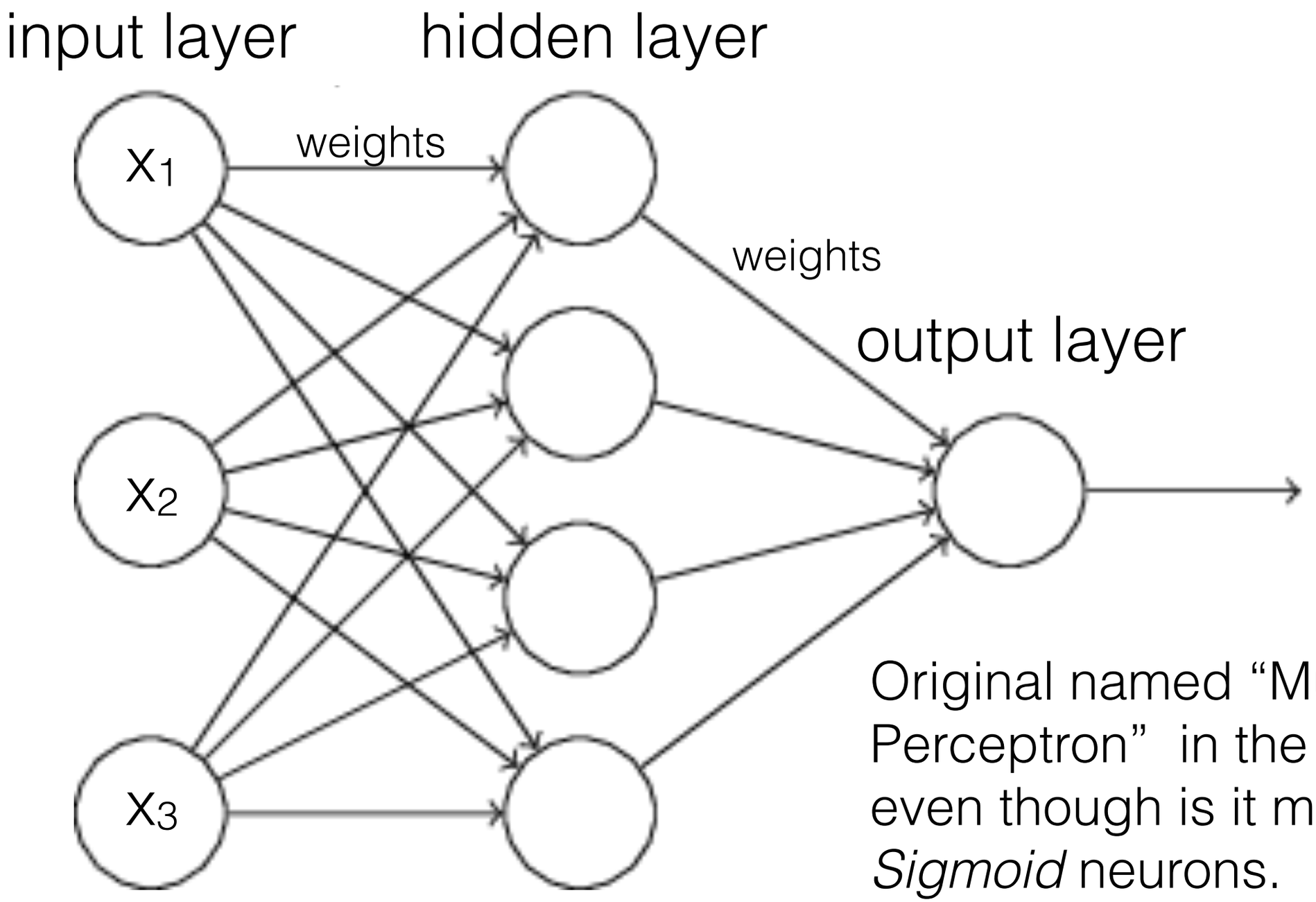
~/gradient descent.ipynb



Our first multilayer neural network

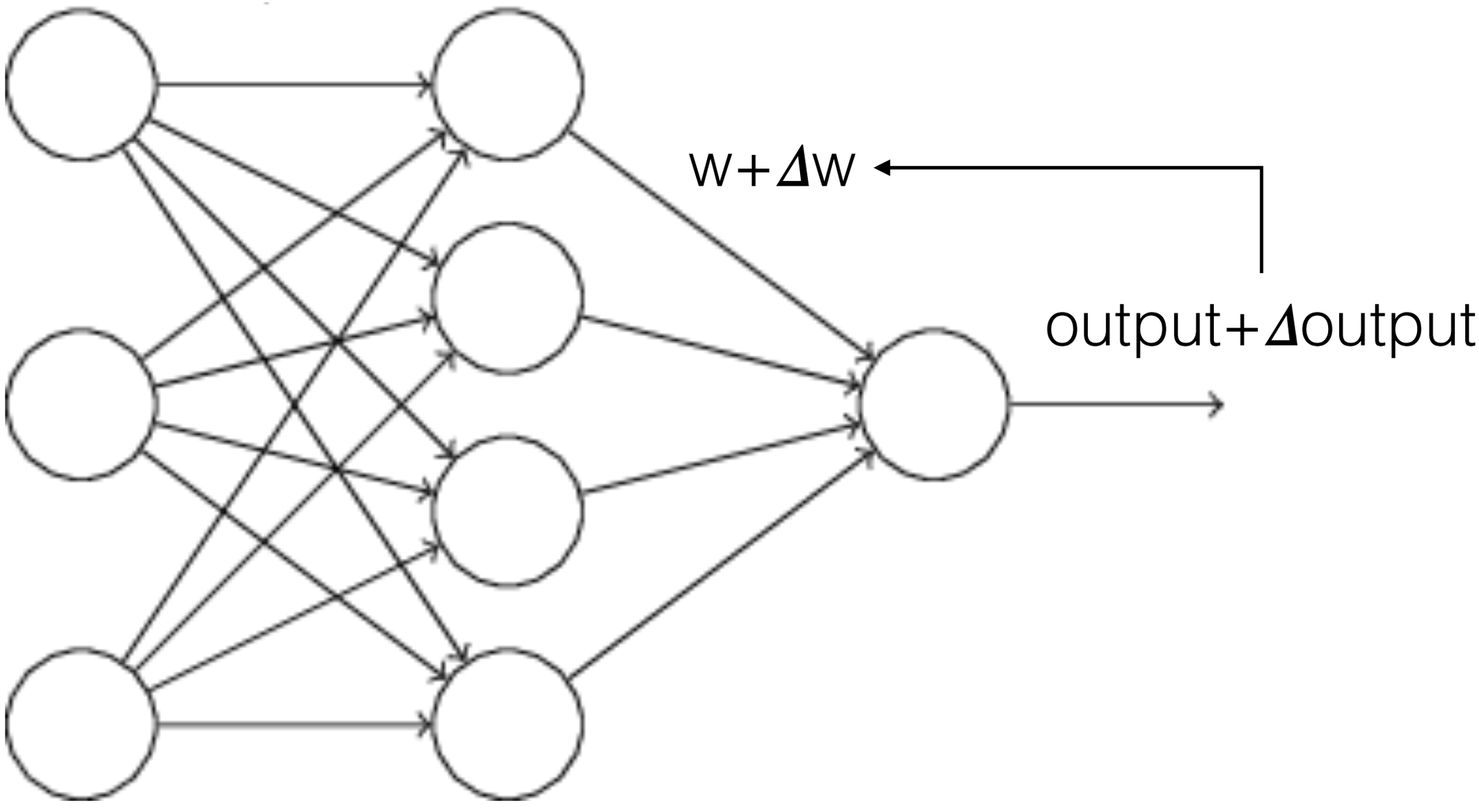


Our first multilayer neural network

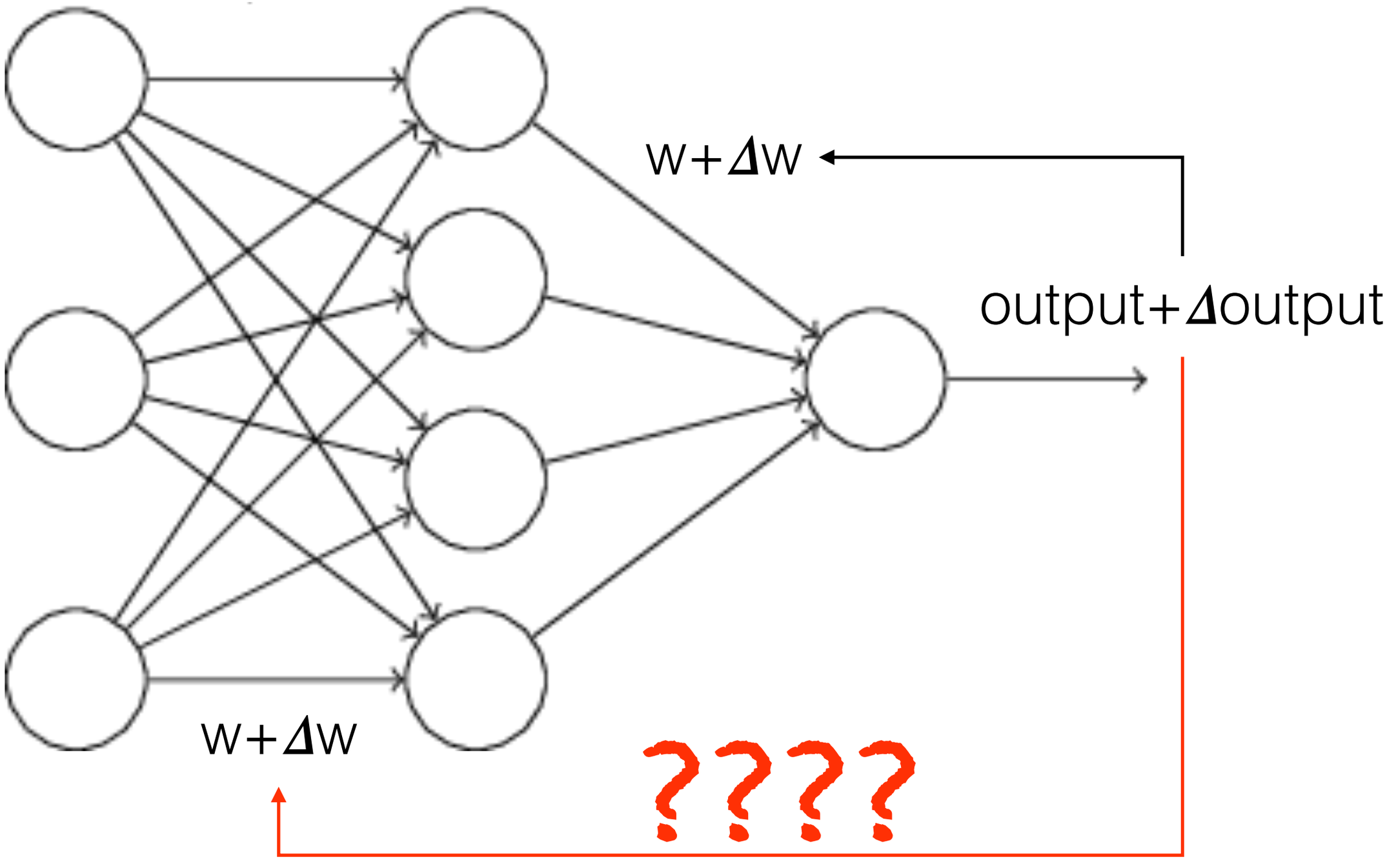


Original named “Multilayer Perceptron” in the 1980s, even though is it made from *Sigmoid* neurons.

How do we train a multi layer network?



How do we train a multi layer network?



How do we train a multi layer network?

+Δoutput

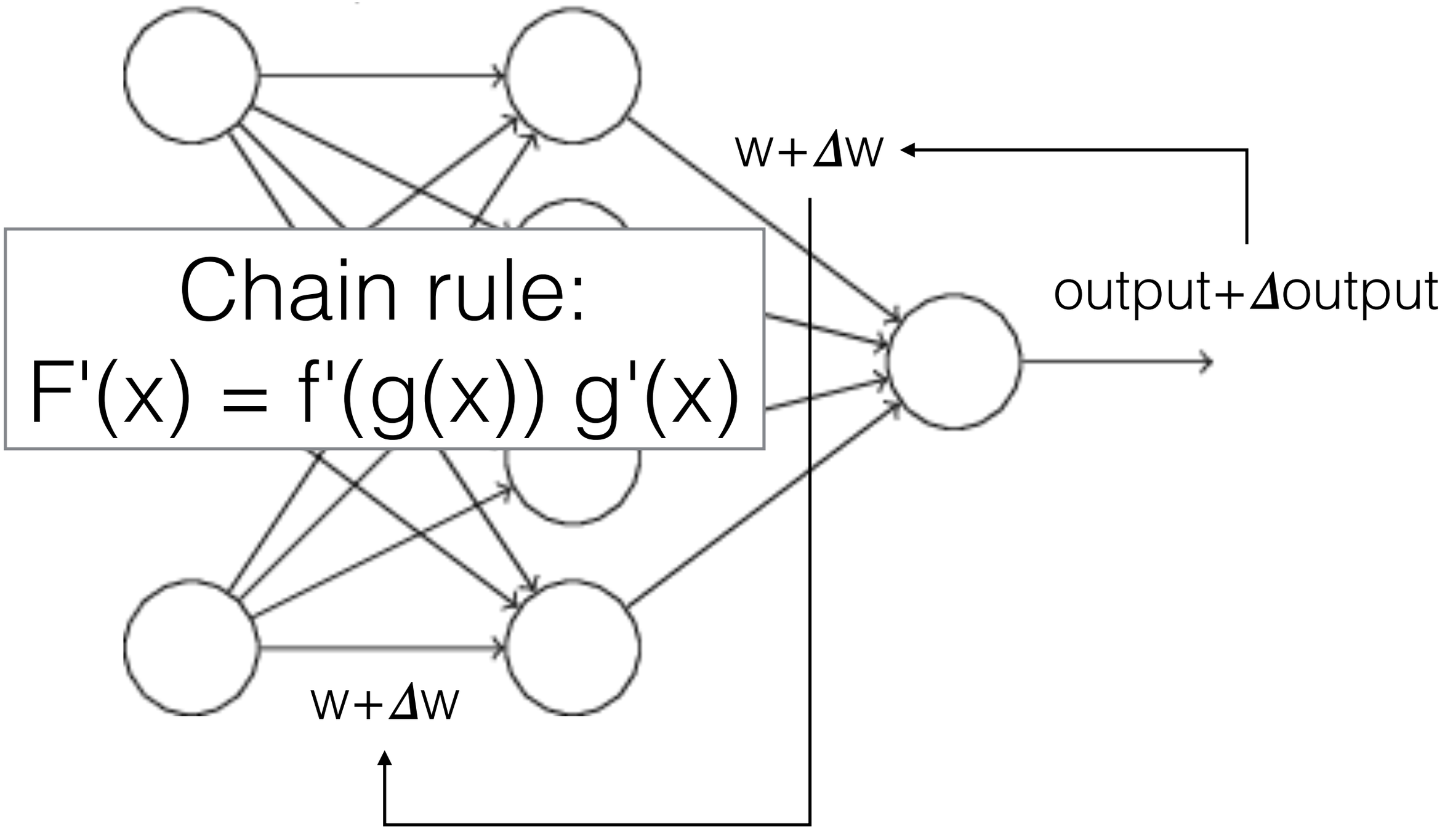
How do we train a multi layer network?

+Δoutput

Backpropagation

or in long: Backward propagation of errors (1970/86)

Backpropagation



So how get NNs trained?

1. Forward path to get predictions
2. Computation of the loss (error)
3. Backward pass to propagate the errors to the weights
4. Update of the weights

Gradient Descent's caveat

Mean Squared Error

$$C(w,b) \equiv \frac{1}{2n} \sum_x \|y(x) - a\|^2.$$

C = cost function (here Mean squared error)
w = weights
b = bias

x = input
y(x) = desired output
a = output
||v|| = length of vector
n = number of sample

Mean Squared Error

$$C(w,b) \equiv \frac{1}{2n} \sum_x \|y(x) - a\|^2.$$

If the number of training vectors (n) is large, the sum is expensive to calculate.

C = Mean squared error)

w = weights

b = bias

y(x) = desired output

a = output

||v|| = length of vector

n = number of sample

Stochastic Gradient Descent (SGD) using batches

Problem: Calculating the cost functions for the complete training data is expensive.

Solution: Split the training set into random batches and for each step only compute the gradient using one batch. Iterating through all batches is called an *epoch*.

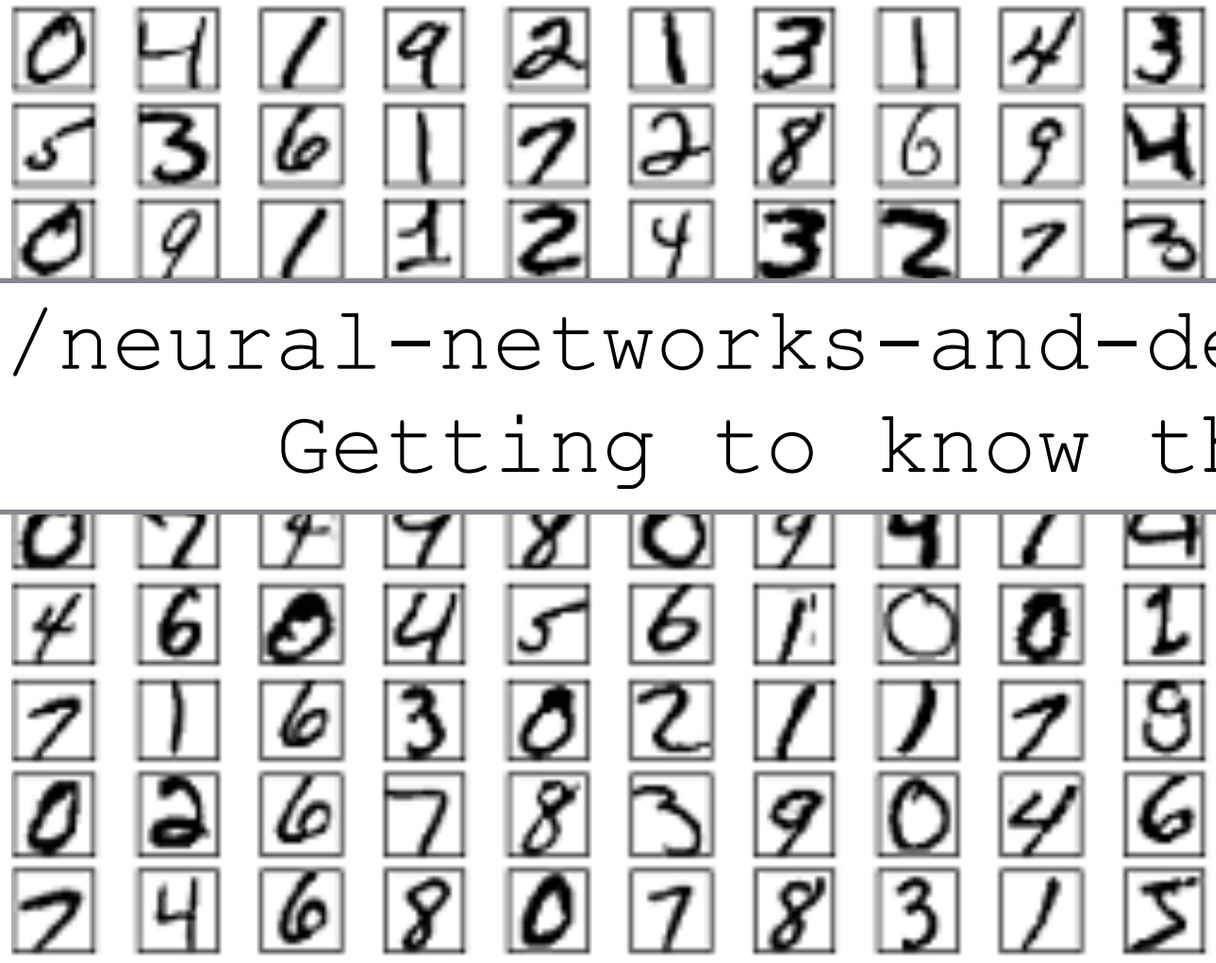
Analogy: Carrying out a political survey instead of holding a general election

Back to the application

MNIST 1998

- Hand written digits (0-9)
- Collected by the United States' National Institute of Standards and Technology, NIST
- Pre segmented
- 28x28 pixel in size
- 60k train, 10k test
- ('M' stands for modified)

MNIST 1998

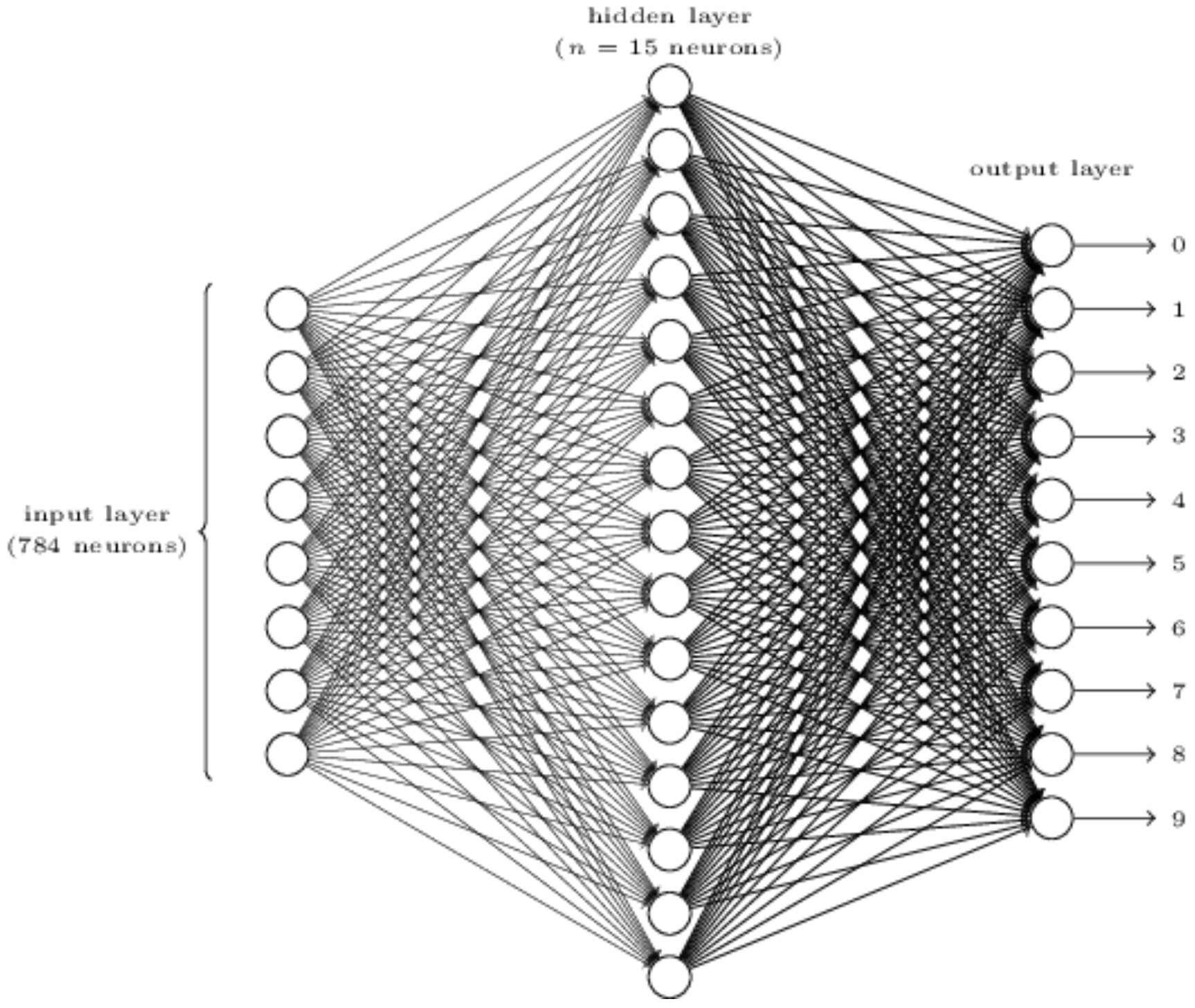


- Hand written digits (0-9)
- Collected by the United States' National Institute of

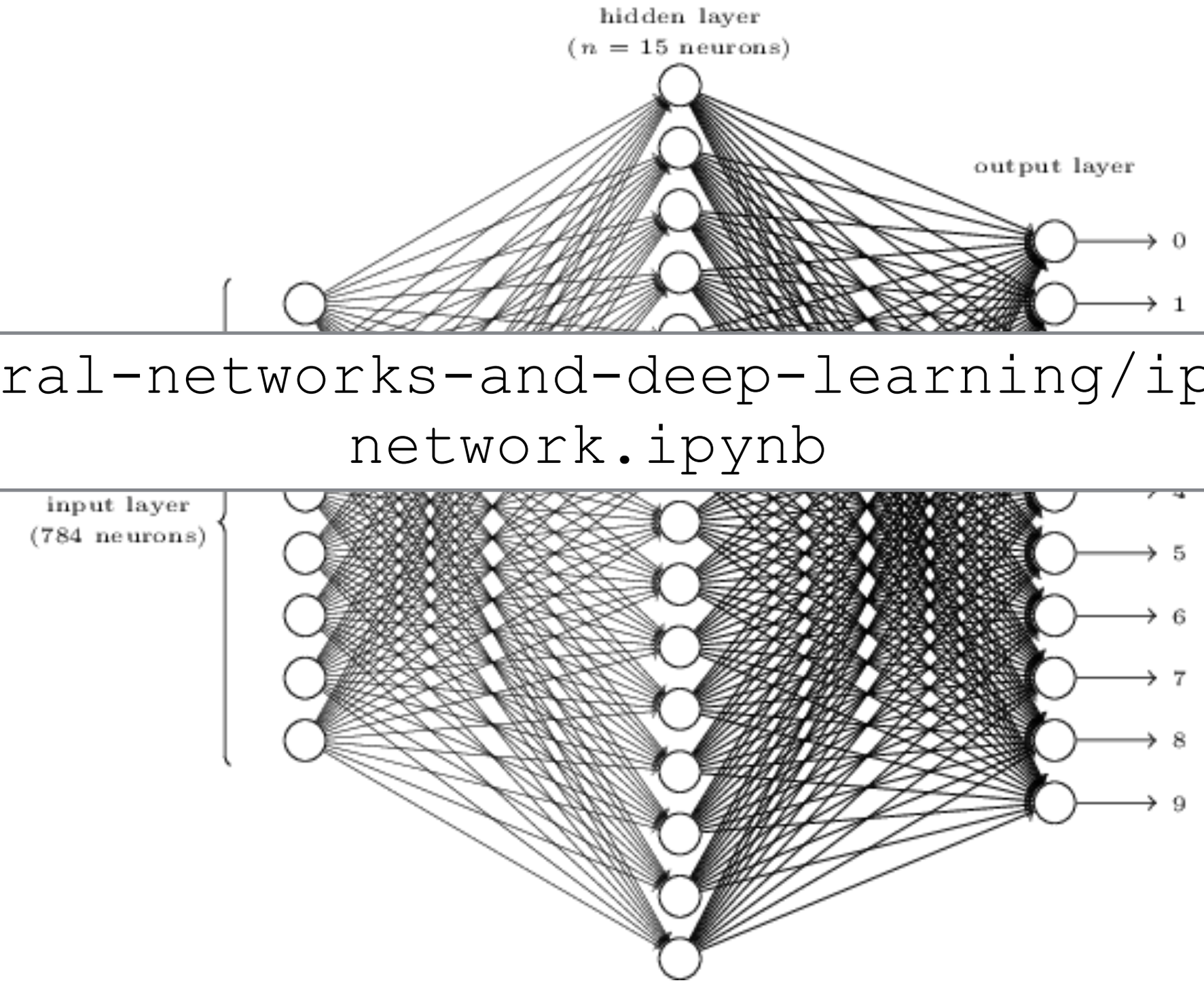
~/neural-networks-and-deep-learning/ipynb/
Getting to know the data.ipynb

- Pre segmented
- 28x28 pixel in size
- 60k train, 10k test
- ('M' stands for modified)

A neural network for MNIST



A neural network for MNIST




```
~/neural-networks-and-deep-learning/ipynb/
network.ipynb
```

`Advanced` Neural Network techniques

A closer look at Mean Squared Error Loss

Mean Square Error loss function:

$$C = \frac{(y-a)^2}{2},$$

MSE's derivatives:

$$\frac{\partial C}{\partial w} = (a-y)\sigma'(z)x = a\sigma'(z)$$

$$\frac{\partial C}{\partial b} = (a-y)\sigma'(z) = a\sigma'(z),$$

with $a=\sigma(z)$ and $z=wx+b$

A closer look at Mean Squared Error Loss

Mean Square Error loss function:

$$C = \frac{(y-a)^2}{2},$$

MSE's derivatives:

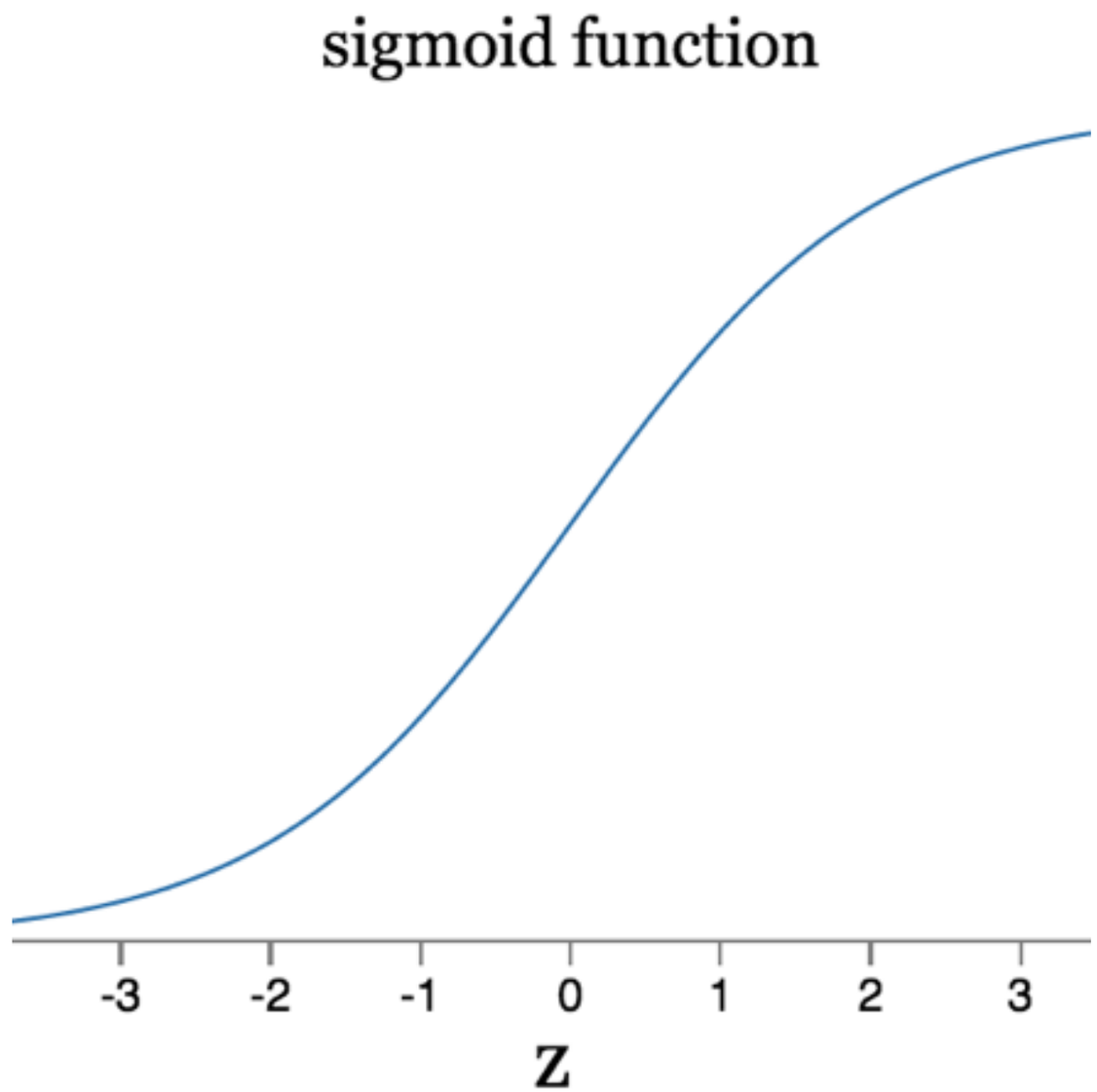
$$\frac{\partial C}{\partial w} = (a-y)\sigma'(z)x = a\sigma'(z)$$

$$\frac{\partial C}{\partial b} = (a-y)\sigma'(z) = a\sigma'(z),$$

with $a=\sigma(z)$ and $z=wx+b$

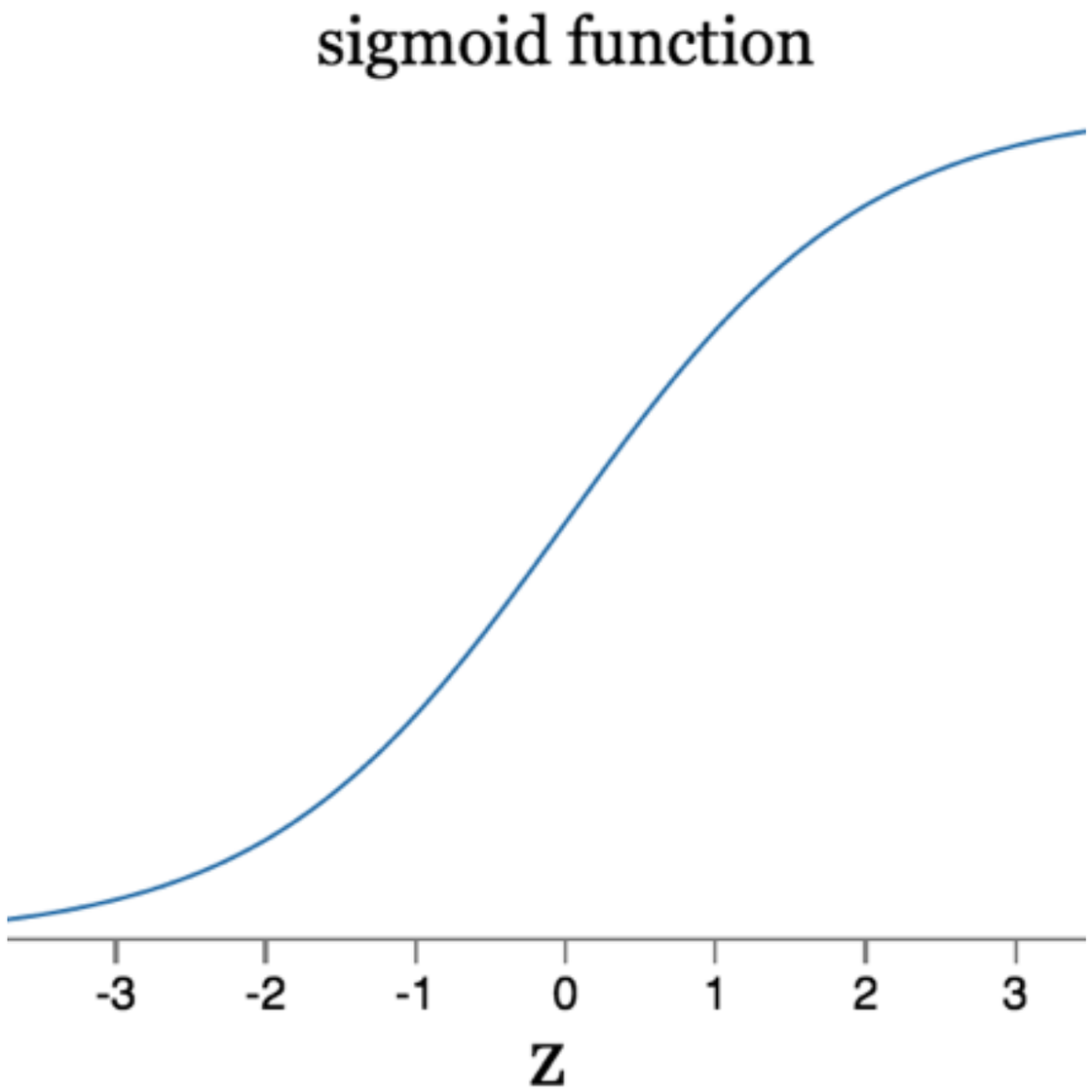
So, what happens when one node is very wrong?

How does the derivative of the sigmoid function looks like?



A better loss function

- **Problem:** Slow down of learning for extreme errors
- **Reason:** Small derivatives in extremes, see example on the right
- **Solution:** Use different loss function like Cross Entropy.



Cross entropy

Definition:

$$C = -\frac{1}{n} \sum_x [y \ln a + (1-y) \ln(1-a)] ,$$

Derivatives:

$$\frac{\partial C}{\partial w_j} = \frac{1}{n} \sum_x x_j (\sigma(z) - y).$$

$$\frac{\partial C}{\partial b} = \frac{1}{n} \sum_x (\sigma(z) - y).$$

Cross entropy

Definition:

$$C = -\frac{1}{n} \sum [y \ln a + (1 - y) \ln(1 - a)] ,$$

(Cross) **entropy** is a concept from Information Theory and is a measure for “surprise”.

Deriv

$$\frac{\partial C}{\partial b} = \frac{1}{n} \sum_x (\sigma(z) - y).$$

SOFTMAX
Activation function

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$$

for $j = 1, \ldots, K$.

Turns the output into a probability distribution.

- All values between 0 and 1
- Sum of all values == 1

SOFTMAX
Activation function

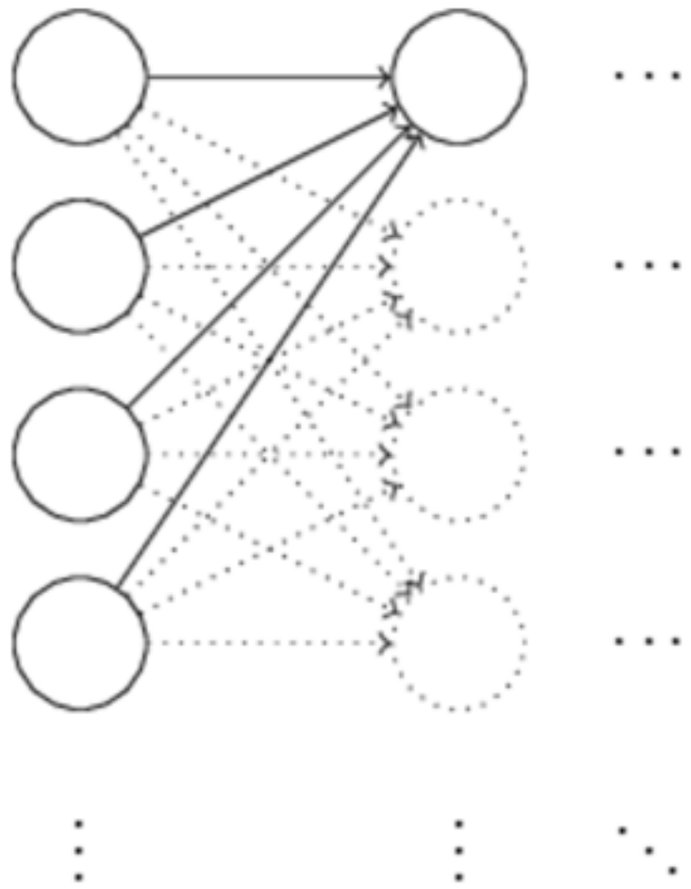
$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$$

for $j = 1, \ldots, K$.

[1		[0.024
2		0.064
3		0.175
4	=>	**0.475**
1		0.024
2		0.064
3]		0.175]

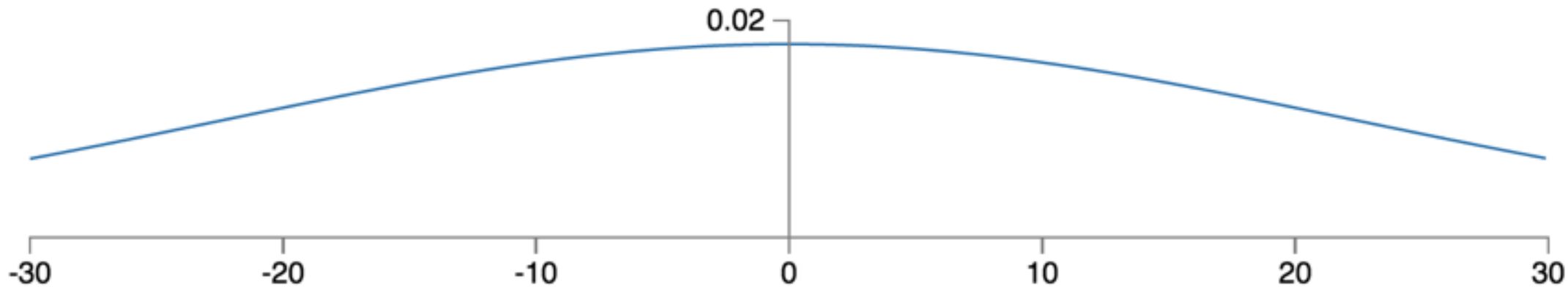
Talking about weight initialization

- Many inputs into one neuron
- Each weight is initialized by a independent Gaussian random variables, normalized to have mean 0 and standard deviation 1.



Talking about weight initialization

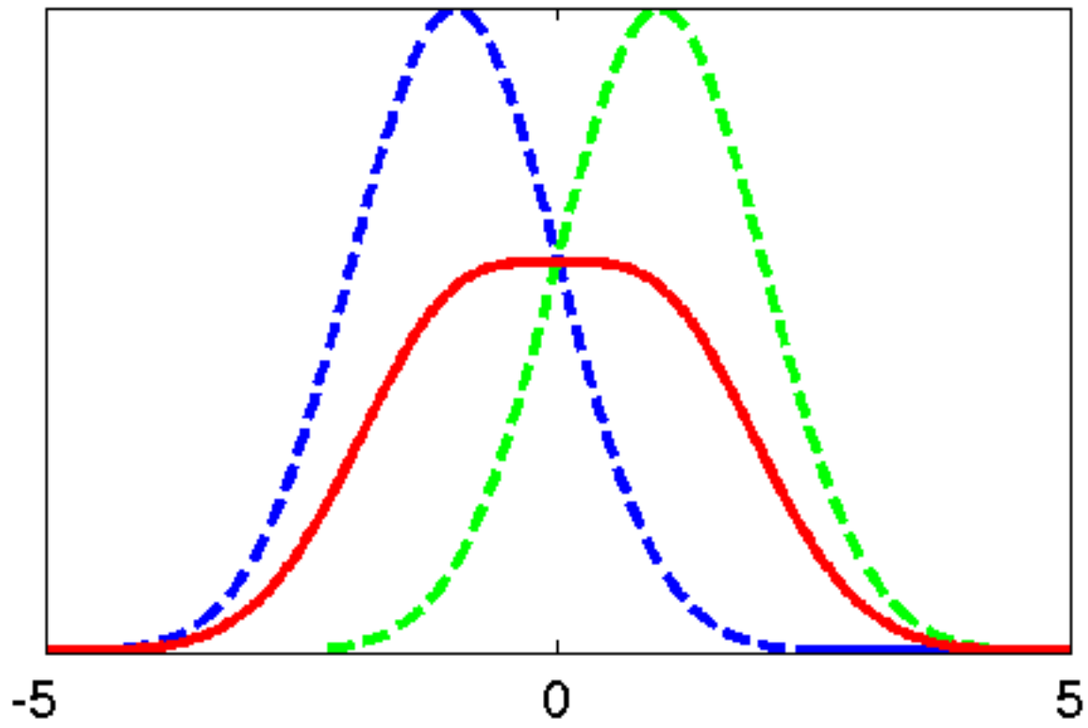
➡ Effect is activations of the neuron is very big.



Here the distribution for 500 non zero inputs

Talking about weight initialization

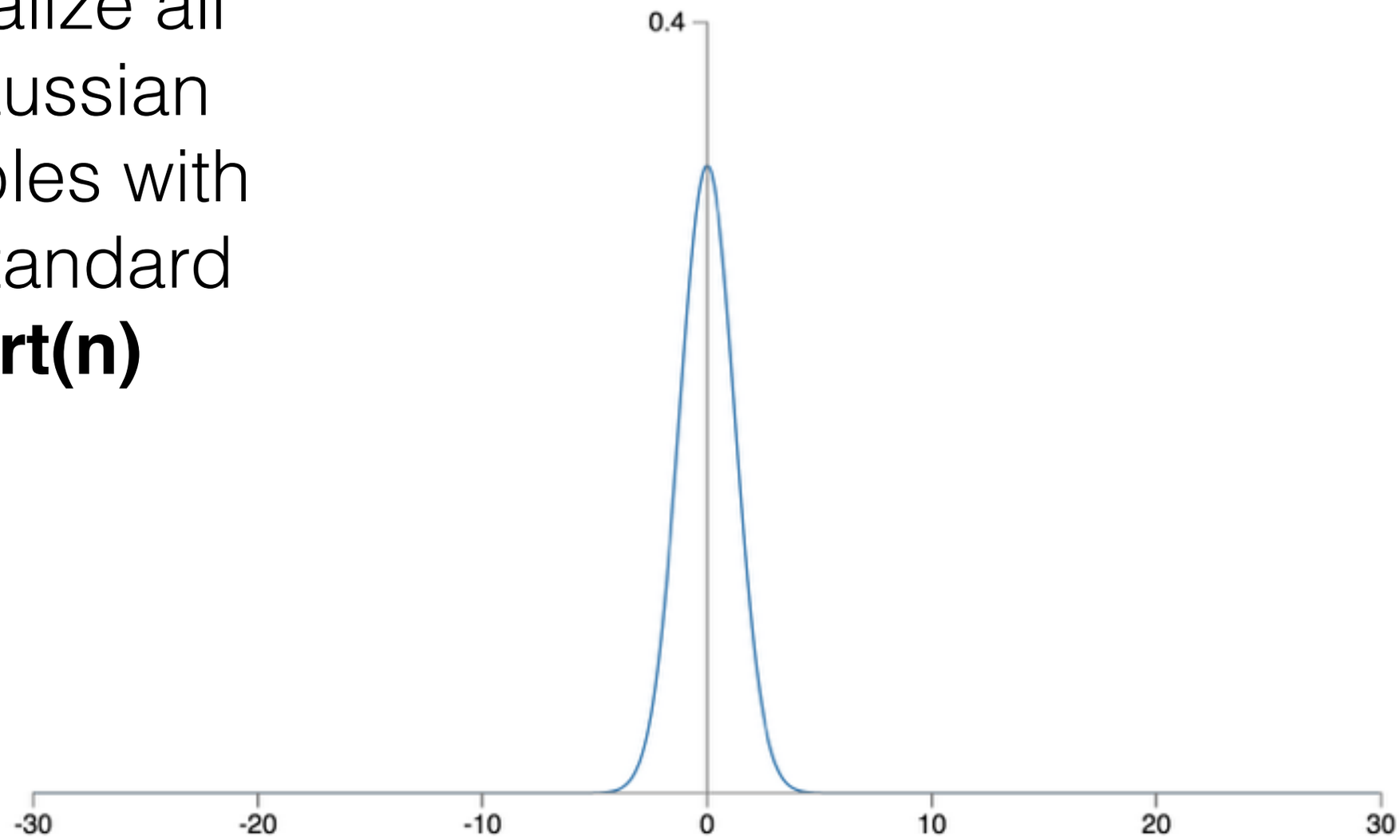
But why?



https://memming.wordpress.com/2012/08/28/mixture-of-point-processes/

Better weight initialization

Solution: Initialize all weights as Gaussian random variables with mean 0 and standard deviation **1/sqrt(n)**



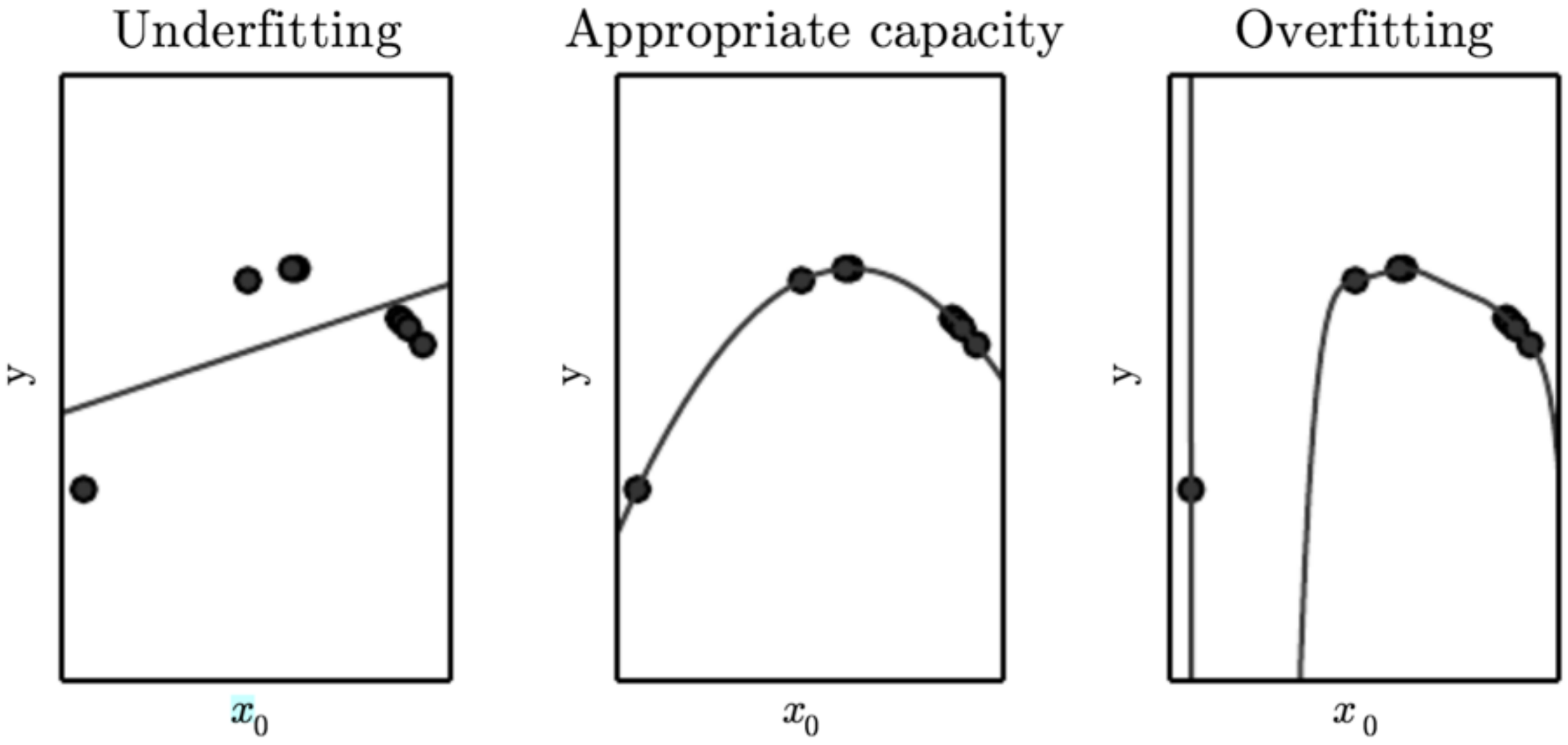
ML basics refresher

Q: What is *Overfitting*?

A: "*In overfitting, a statistical model describes random error or noise instead of the underlying relationship. Overfitting occurs when a model is excessively complex, such as having too many parameters relative to the number of observations.*"

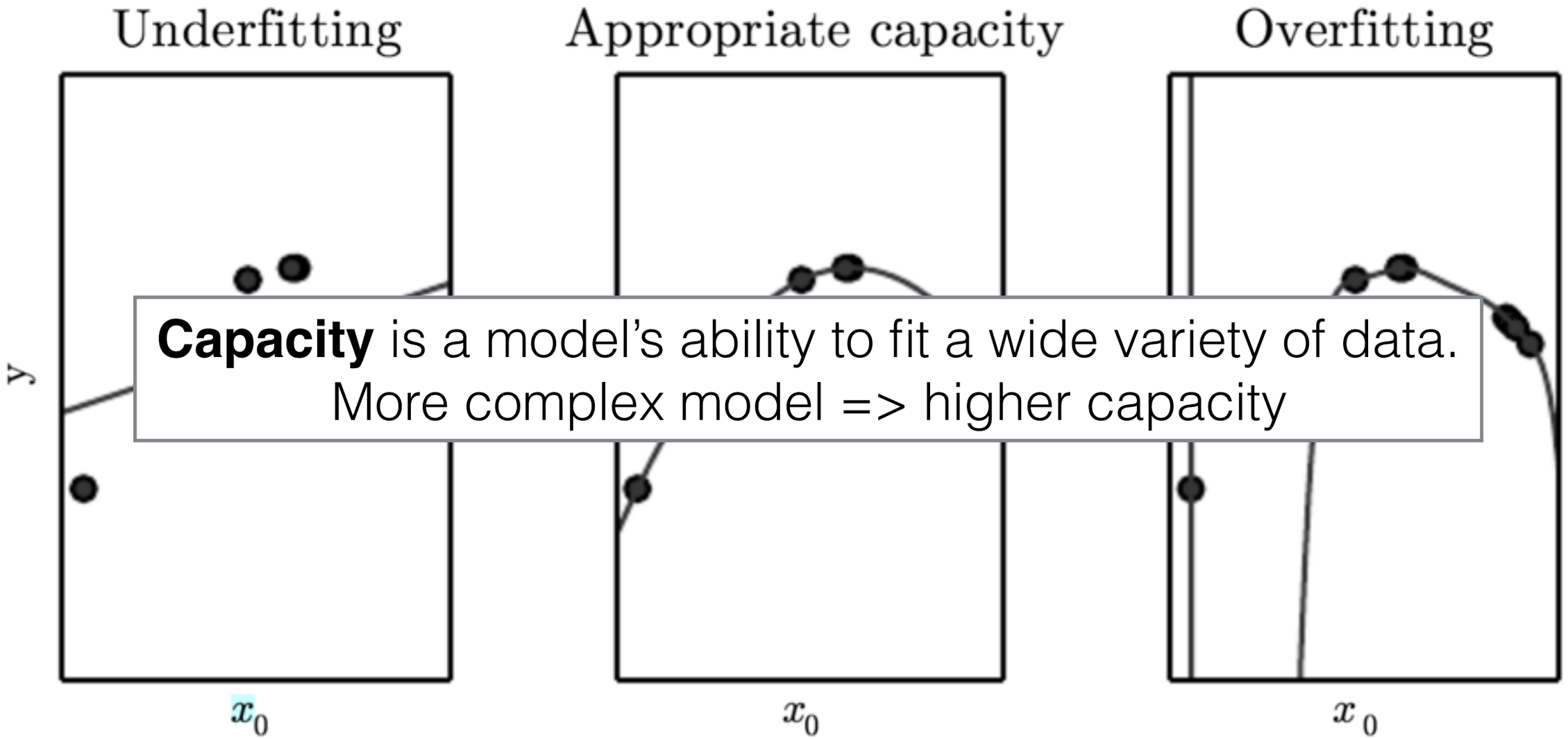
en.wikipedia.org/wiki/Overfitting

ML basics refresher



www.deeplearningbook.org/

ML basics refresher



Capacity is a model's ability to fit a wide variety of data.
More complex model => higher capacity

www.deeplearningbook.org/

So what is the capacity of a deep NN?

So what is the capacity of a deep NN?

- It is very high.
- All datasets can be overfit with enough time and processing power if the network is large enough!

=> Use *regularization* to introducing additional “information” / constraints to prevent overfitting.

Preventing overfitting with L2 regularization

- big weights are sensible to noise
- to keep them small we penalize big weights in the cost function. This can be done with:

L2 regularization aka *weight decay*

$$C = C_0 + \boxed{\frac{\lambda}{2n} \sum_w w^2}$$

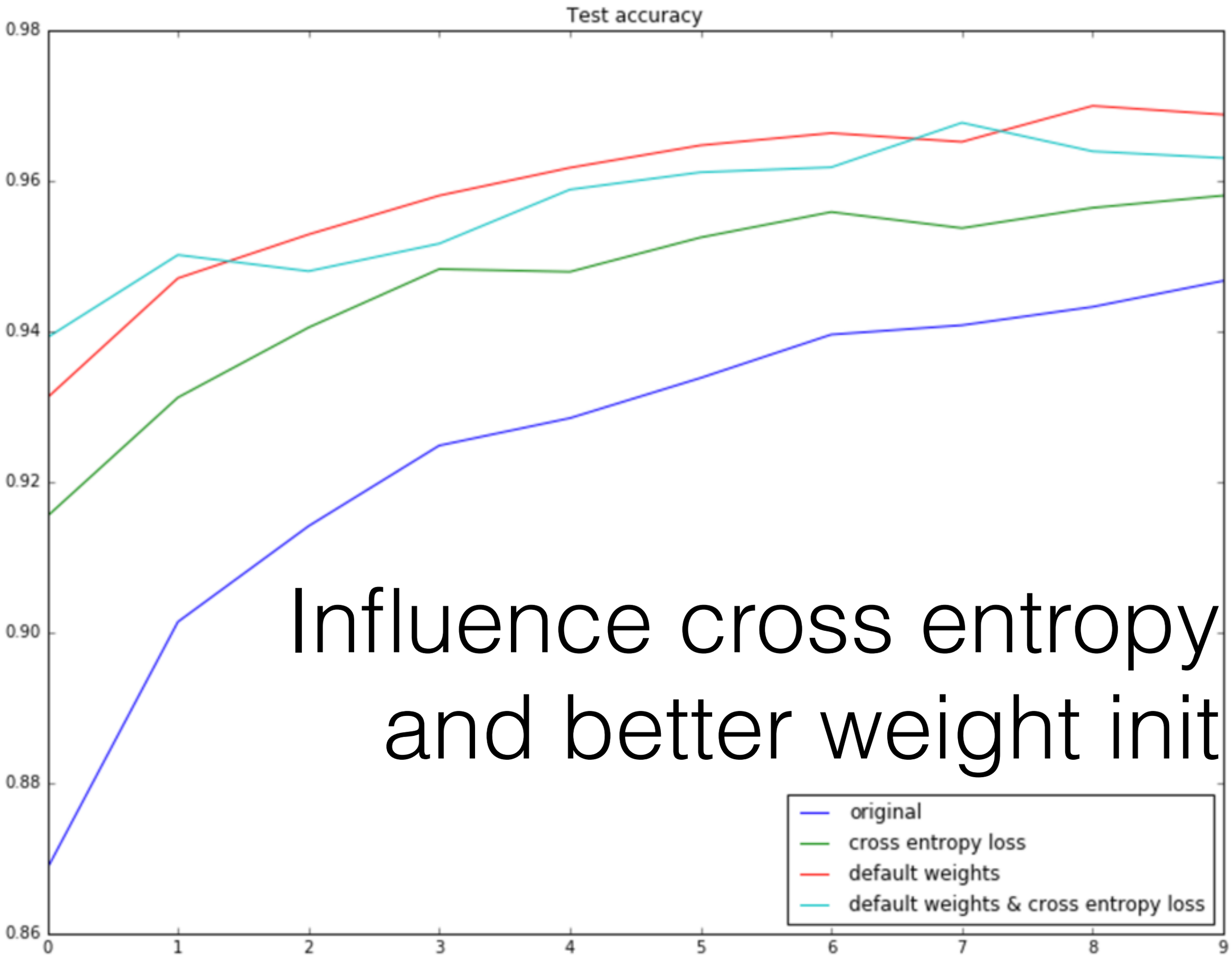
Preventing overfitting with L2 regularization

- big weights are sensible to noise
- to keep them small we penalize big weights in the cost function. This can be done with:

L2 regularization aka *weight decay*

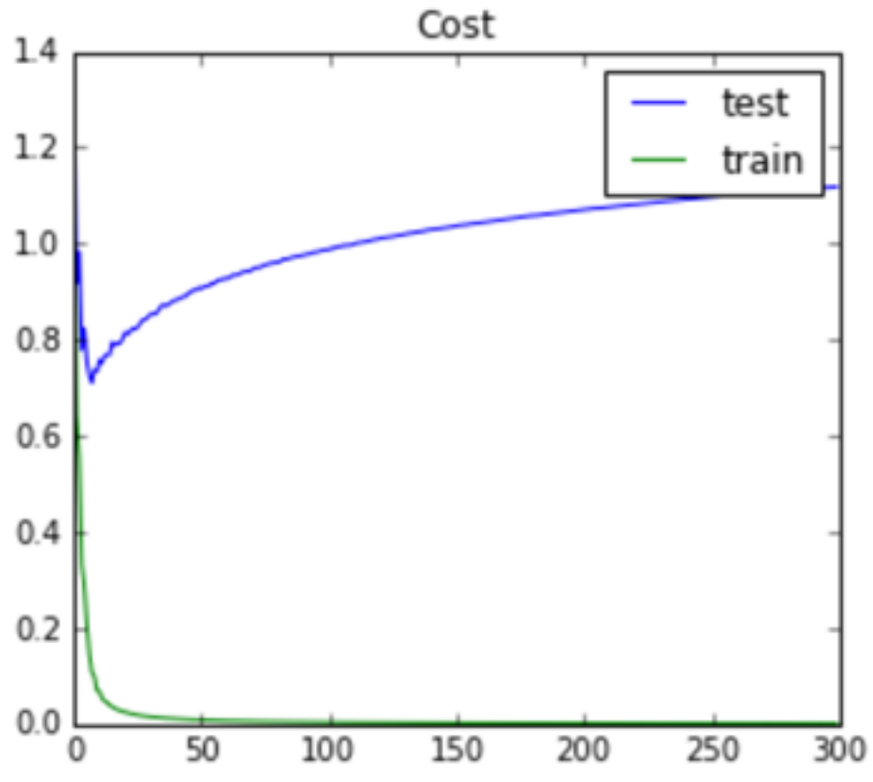
$$C = C_0 + \frac{\lambda}{2n} \sum_w w^2$$

Don't forget to adjust λ for n, the number of training images

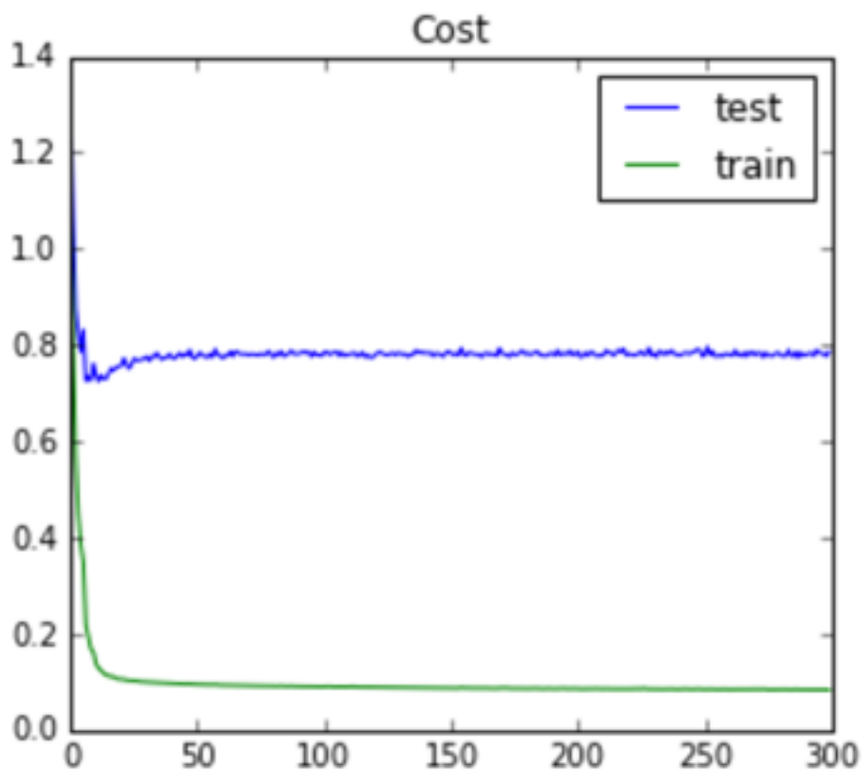


Influence cross entropy and better weight init

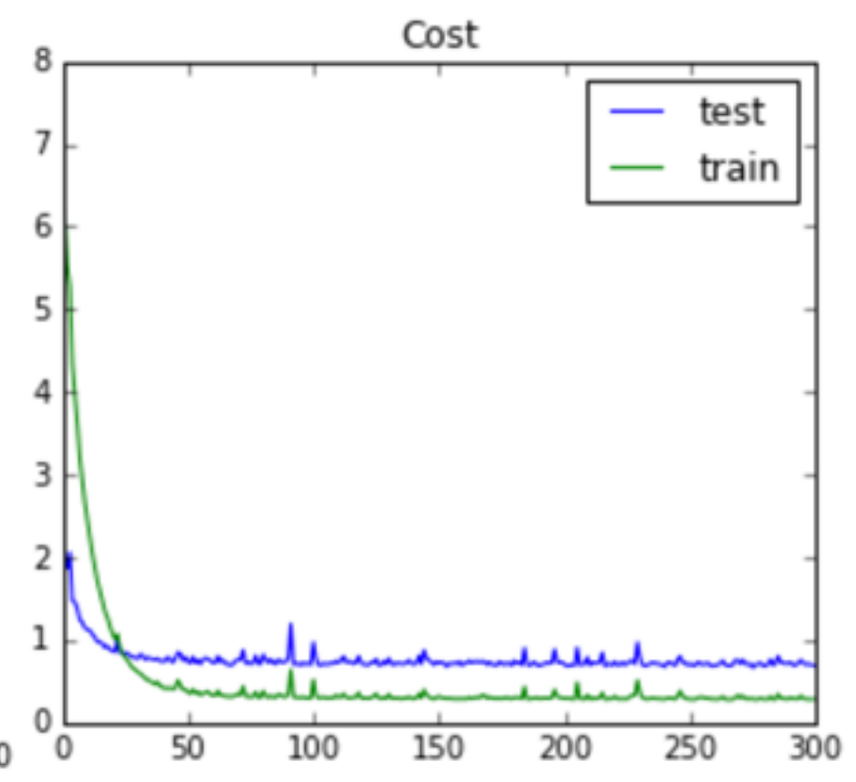
Influence regularization



no regularization *



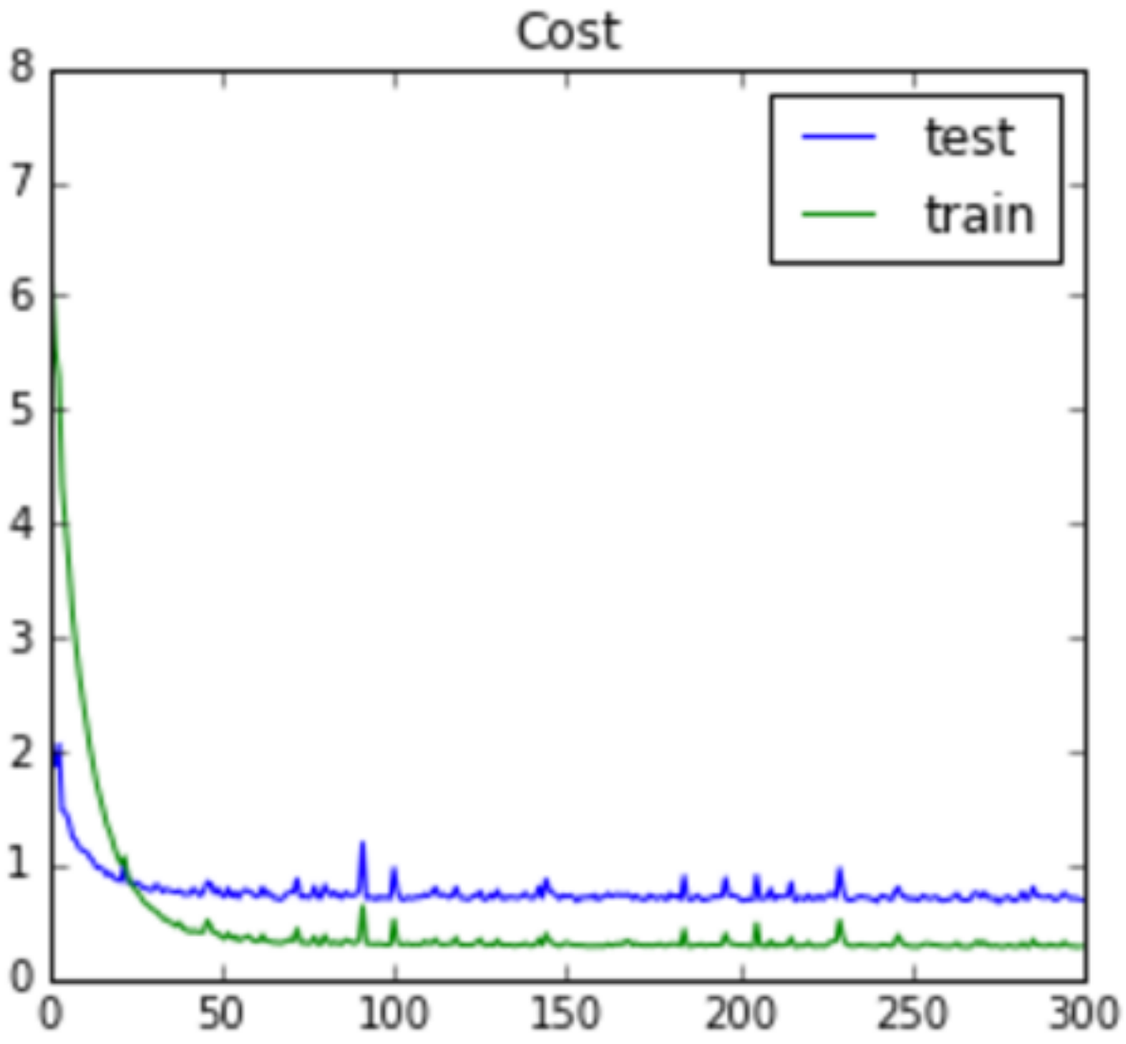
regularization = 0.1 *



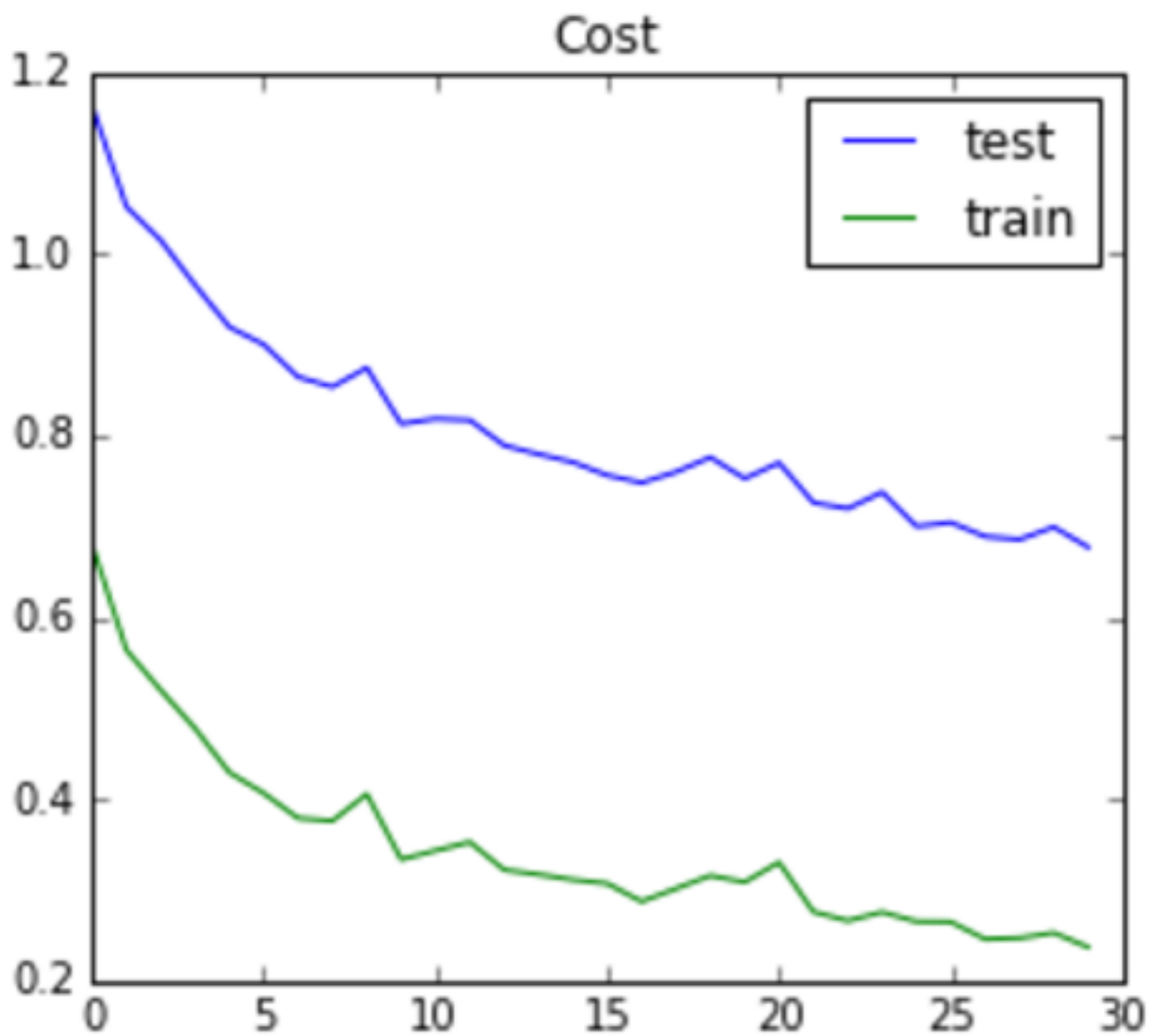
regularization = 0.5 *

* using only 1000 training examples for speed reasons

Influence regularization



1000 training examples



50000 training examples

both regularization = 0.5